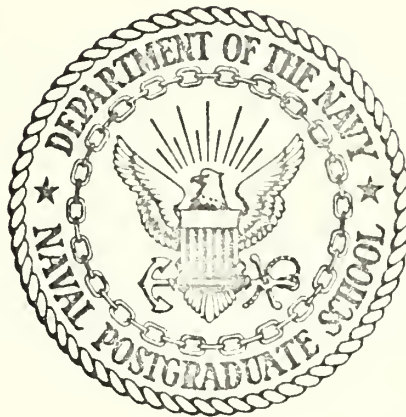


INTERACTIVE LOGIC LABORATORY

Robert Lee Johnson

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

INTERACTIVE LOGIC LABORATORY

by

Robert Lee Johnson, Jr.

Thesis Advisor:

George A. Rahe

June 1972

Approved for public release; distribution unlimited.

INTERACTIVE LOGIC LABORATORY

by

Robert Lee Johnson, Jr.
Lieutenant, United States Navy
B.S., Naval Postgraduate School, 1971

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

ABSTRACT

This thesis documents an investigation into the use of a computer graphics terminal to demonstrate the basic concepts of logical design. The areas of computer-assisted instruction, computer graphics, and computer-aided design are reviewed prior to the discussion of the creation of the INTERACTIVE LOGIC LABORATORY. The program is implemented on the Adage Graphics Terminal - 10 (AGT-10) of the Naval Postgraduate School Computer Laboratory.

The main emphasis of the program discussion is on the degree of interaction achieved by the program and its possible use as a learning aid for students of basic logical design courses. A bipartite graph is used to depict the network topology of the logic circuit and the program is quite successful in the simulation of simple logic circuits.

TABLE OF CONTENTS

I.	INTRODUCTION (Interactive Logic Laboratory)-----	5
II.	BACKGROUND-----	8
	A. EDUCATIONAL AIDS-----	8
	B. SUMMARY OF PREVIOUS RESEARCH-----	9
	1. Computer-Aided Design-----	10
	2. Circuit Design Programs-----	12
	3. CIRCAL: On-Line Circuit Design-----	13
	4. Computer-Assisted Instruction-----	14
	5. Basic Logical Design-----	16
III.	PROGRAM IMPLEMENTATION-----	18
	A. COMPUTER SELECTION-----	18
	B. FLOW-OF-CONTROL-----	20
	C. DATA STRUCTURE-----	21
	D. MAIN PROGRAM ROUTINES-----	24
	1. Define Mode-----	25
	2. Connect Mode-----	27
	3. Analysis Mode-----	28
	a. Circuit Analysis-----	29
	b. User Input-----	33
	c. Circuit Response-----	34
IV.	PROGRAM EVALUATION-----	38
	A. SAMPLE TERMINAL SESSION-----	38
	B. RESULTS-----	44
	C. EXTENSIONS-----	45
	D. RECOMMENDATIONS AND CONCLUSIONS-----	46

APPENDIX A-----	47
COMPUTER PROGRAM-----	50
LIST OF REFERENCES-----	124
INITIAL DISTRIBUTION LIST-----	126
FORM DD 1473-----	127

I. INTRODUCTION

The value of laboratory work has long been recognized in the physical sciences. This most important adjunct to the learning process has both the demonstrative power of presenting classroom concepts in a real-world setting as well as allowing the student to learn by doing. The experience gained by actual experimentation in the laboratory serves the dual purpose of aiding the learning process and preparing the student for more rigorous research in later work.

If Alfred North Whitehead's triad of learning -- a stage of romance, a state of precision, a stage of generalization [ref. 1] - is accepted, the value of the laboratory becomes particularly evident. Thought-provoking demonstrations, quite easily presented in the laboratory environment, aid immensely in arousing student interest in a topic. The ability for precise analysis can be developed by student experimentation. Thus leading to a lesser reliance on previously learned specifics, a better understanding of the fundamental concept and the ability to extrapolate to its implication.

The value of laboratory experience is also well appreciated in the computer sciences as evidenced by the huge investment institutions of higher learning have made in computer facilities, both for actual research and student experimentation. For instance, a dual-processor IBM Model 360-67 is installed at the Naval Postgraduate School and this facility is utilized for both administrative work as well as extensive student and faculty research. Another computer facility implemented at the school is a hybrid computer comprised of an XDS-9300 processor

interfaced with a Comcor CI-5000 analog computer and two Adage (AGT-10) Graphics terminals.

It is impossible for a student at the Naval Postgraduate School to complete any of the various courses of study without some degree of exposure to one of these computation facilities. Yet, none of these computer installations is currently adapted for demonstration and experimentation in one of the most basic areas of computer science - logical design. Various logic demonstrators have been marketed, in fact the Comcor CI-5000 has the capability of being used as a logic demonstrator. However, the use of this analog computer for demonstration of basic logic design concepts has not been explored due to the high cost associated with this implementation.

The lack of adequate laboratory facilities in a basic logical design course has been apparent to several of the members of the faculty at the Naval Postgraduate School. Various approaches to the problem have been tried. One professor assigned work on the CDC-160 which involved using that machine's bit manipulative capabilities, both logical and arithmetic, to simulate the various logical operations discussed in class. A more extensive effort to implement a meaningful laboratory environment for basic logical design consisted of a computer program written for the IBM-360 which performed detailed digital machine simulation at the bit-handling level through control unit level. This program [ref. 2] was designed to provide the student with an operating model of the digital computer capable of both demonstrating computer operations and allowing realistic experiments in logical design.

While the program was a significant improvement over any previous laboratory capability, it still was lacking in one important respect -- the simulation of a hardware - oriented subject was done using software.

The published article recognized "The need for an early course introducing the digital computer as a hardware system." [2] Yet student programming ability in a higher level language was presupposed as a requirement for use of the digital machine simulation program.

With this background it was desired to use existing facilities at the Naval Postgraduate School to investigate the implementation of a meaningful instructional vehicle for presenting the basic concepts of logical design which did not require student programming ability in a higher-level language. The criteria by which the effectiveness of the implementation would be measured were:

- 1) The ability of the program to clearly demonstrate, in familiar terms, the basic concepts of logical design.
- 2) The ability of the program to allow for student experimentation.
- 3) Ability of the program to be utilized by a large number of students with minimal instruction and minimal interference with other computer users.

Thus the problem consisted of investigating the various requirements of such a program, determining a suitable set of equipment for implementation, and finally creating such a program to demonstrate the feasibility of the concept. The intent of this thesis was not to actually implement the program at the classroom level, but to investigate the construction of such a program in sufficient depth to allow subsequent extension of the concepts considered in the thesis to a general classroom implementation.

II. BACKGROUND

A. EDUCATIONAL AIDS

Many different educational aids have been developed and marketed over the past few years. The teaching aids committee of The American Society for Engineering Education (ASEE) was established to review and recommend various educational aids to institutions of higher learning. In doing so, they established definitions of various types of educational aids which were considered pertinent to this study. In particular, they subsumed the general term "educational aid" into the more comprehensive categories of teaching aids, learning aids and training aids.

Of special importance to this project was the ASEE definition of a learning aid as "a device that helps in the understanding of a fundamental of engineering or science and which creates within the student a desire to pose and solve a problem by an application of the fundamental involved."

Of the three categories defined, the committee stressed most heavily the importance of learning aids in the educational process, stating in part:

"The old adage, 'Experience is the best teacher' might be rephrased thus, 'Experience is the best way to learn.' All of the experimental evidence from educational psychology tends to substantiate this concept... student participation, either purely mental or a combination of physical and mental, is the key to successful efficient learning."

Hence in the construction of the proposed educational aid, it was mandatory that it permit direct student experience in the actual design

of computer logic circuits. This would most profitably be done within a framework that would enable the student to construct a logic circuit in familiar form using standard symbology. Further experience would be obtained by allowing student control of the inputs to the constructed circuit and observation of the responses of the circuit to these inputs. The major emphasis being on student control of variables and analysis of results.

B. SUMMARY OF PREVIOUS RESEARCH

The above requirements posed by an investigation of the basic considerations in the design of an effective educational aid when considered together with the requirement for clarity and ease of use indicated a graphical approach to the problem. The expression, "One picture is worth a thousand words" has been used (almost too) extensively to characterize the value of graphical display, but nonetheless underscores the impact of this means of presenting information. Additionally, the requirements for student interaction with and control of the learning aid strongly suggested that it could best be constructed by using a digital computer with graphical input and output capability. The computer would be programmed to accept and display student inputs, perform analysis on these inputs, and display the results of the student's circuit design efforts.

Hence the fields of computer graphics in general and computer-aided design in particular were investigated as a prelude to construction of the program. The term, "computer graphics," has been defined [ref. 3] as that set of computer techniques and applications wherein data is either presented or accepted by a computer in graphical form. "Computer-aided design" (CAD) as the name implies, specifies the use of a

computer as an assistant in the design of some entity. While computer graphics is a general enough term to encompass all computer systems using graphical display, computer-aided design usually implies a significant degree of man-computer interaction in the symbiotic relationship perceived by Licklider [ref. 4]. The designer specifies his ideas to the computer in graphical form, uses the computer's significant computational ability to perform some analysis of the design, possibly modifying the design for re-analysis. While the purpose of the proposed program did not include its specific use as a design aid for actually fabricating logic circuits, it was felt that the computer-aided design approach would be most successful in exposing the student to design concepts as applied to logic circuits.

1. Computer-Aided Design (CAD)

Early research in the field of computer-aided design centered at the Massachusetts Institute of Technology. Professor Steven A. Coons of the mechanical engineering department at that institution published a paper in the early 1960's outlining the requirements for a computer-aided design (CAD) system [ref. 5]. The paper traced CAD development from its genesis in the Automatic Programmed Tool System to the level of sophistication displayed by Sutherland's SKETCHPAD [ref. 6].

In examination of the design process itself, Coons saw "a few engineers performing highly creative tasks at the beginning, coupled with a very large number of draftsmen and technicians, who perform relatively uncreative tasks over a fairly long period of time." [5] He further envisioned that this process could be vastly improved by using a computer with a graphical capability to accept, interpret, and remember shape descriptive information. Additionally, the computer system

must have the ability to perform the mathematical analysis necessary to evaluate the design with respect to predetermined objectives.

In support of his vision of improvement in the design process by computer, Coons enumerated several CAD system benefits:

- 1) Emphasis on interaction and inter-communication between design users.
- 2) Dynamic display of time-varying systems.
- 3) Use of more accurate mathematical models, allowed by increased computational power.
- 4) Exponential design rate, with subelements of design saved by computer.
- 5) Use of the same basic structure by different design disciplines.

No discussion of computer graphics or computer-aided design would be complete without reference to the pioneering work of I. E. Sutherland, also at MIT. The SKETCHPAD system [6] was the first to demonstrate the effective use of an interactive display console to accept inputs and display outputs in graphical form and control the sequence of program execution. Sutherland's program was built around a powerful data structure which allowed for representation of display elements, labeling of various parts of the display with alphanumerics, and representation of display topology. Analysis was accomplished in the program by the use of mathematical conditions (called "constraints,") on parts of the drawing. The addition of design constraints as well as geometrical constraints gave SKETCHPAD a significant design capability, although at the time of publication, Sutherland's program had not demonstrated the ability to design electrical circuits.

2. Circuit Design Programs

The view of the nature of the problem as a logic circuit design task indicated a review of extant computer programs constructed for this purpose. Some examples of user-oriented circuit analysis programs reviewed were Electronic Circuit Analysis Program (ECAP) [ref. 7] and Continuous Systems Modeling Program (CSMP) [ref. 8] both IBM circuit analysis applications. Also investigated were Automated Engineering Design of Networks (AEDNET) [ref. 9] and Circuit Analysis (CIRCAL) [ref. 10] by J. Katzenelson and M. L. Dertouzos respectively of MIT. Of most pertinence to the construction of the logic demonstrator was Dertouzos's paper, "Introduction to On-Line Circuit Design." [ref. 11].

In this paper, Dertouzos characterized on-line circuit design as a design dialog with short interaction delays. He further listed various requirements which must be met if the implementation of a circuit design program is to be truly interactive. These include:

- 1) An editing requirement to accomplish inputting of information such as network description, element description, variable values, etc.
- 2) An output requirement to convert computer generated information into a form suitable for transmission to the user.
- 3) A definitional requirement to enable users to build sub-elements of a circuit design to be used in later more complex circuits.
- 4) An informational requirement to provide the system with necessary control information to execute the program.
- 5) A diagnostic requirement to enable the user to discover mistakes in his use of the program or design.

In reference 11, Dertouzos also discussed in detail the internal structure of on-line programs for circuit design emphasizing the need for storing of information and insuring proper information flow between various program segments. The importance of a comprehensive data structure capable of representing the topology of the network was presented along with the benefits of such a data structure. These include:

- 1) Efficient use of storage.
- 2) Efficient application of algorithms.
- 3) Use of operators which are independent of the size and structure of stored information.
- 4) Efficient representation and processing of recursive constructs.

The analysis portion of a typical on-line circuit design program was considered in light of the requirements for interaction, such as premature termination of the analysis by the designer or changes in the course of the analysis designated by the designer. The nature of the on-line approach to circuit design was shown to be an adaptive type of process as opposed to one with predetermined structure.

3. Circal: An On-Line Circuit Design Program

The above requirements for a circuit design program were met in the CIRCAL program implemented at MIT by Dertouzos [10]. Several facets of the program applied directly to the development of the proposed logical design program. The program itself had three distinct versions (CIRCAL-0, CIRCAL-1, CIRCAL-2), each capable of handling an increasingly complex electrical network. The program operated on-line on a modified IBM 7094 under the Project MAC system using either graphical or teletype modes. Of special interest were the use of a grid mesh, superimposed on

the display, with the restriction that circuit elements could lie only on the grid intersections. Connection of individual circuit elements was done using analysis of typewritten commands.

In consideration of the interactional requirements listed above, the program was structured into three main segments. These were:

- 1) A DEFINE mode for circuit elements and waveforms.
- 2) An INPUT/EDIT mode for forming or changing network connections and specifying element values.
- 3) A TEST/OUTPUT mode for observing response of the designed circuit to the specified inputs.

Consideration was given to the importance of the data structure and analysis methods in the overall effectiveness of the system. Also of importance was the observation that "the 'input/edit' and 'define' functions of any on-line circuit design system are in principle independent of the methods used for network analysis." [10].

4. Computer-Assisted Instruction (CAI)

In view of the didactic nature of the proposed program, an investigation of the field of computer-assisted instruction was deemed appropriate. Computer-assisted instruction evolved from the concept of programmed instruction first articulated by S. L. Pressey at Ohio State University in the 1920's. This learner-centered method of instruction presents new information to the student in the form of incremental steps with constant review and testing to reinforce learning. The method did not earn general acceptance until the need for reinforcement in learning was underscored by the research of B. F. Skinner at Harvard University in the middle 1950's. [ref. 12].

G. M. and L. C. Silvern [ref. 13] listed the criteria governing programmed instruction which are now generally accepted as:

- 1) Instruction is provided without the presence of a human instructor.
- 2) The learner progresses at his own rate (conventional group instruction, films, television and other fixed-format media do not satisfy this criterion.)
- 3) Instruction is presented in small incremental steps requiring frequent response by the learner.
- 4) There is a participative, overt interaction or two-way communication between learner and instructional program.
- 5) Learner receives immediate feedback informing him of his progress.
- 6) Reinforcement is used to strengthen learning.

Although the methods of programmed instruction were unusually well suited for computer implementation, their appearance before the general availability of computers to educational institutions lead to initial textual implementation. The original structure of a programmed instructional text was essentially linear in nature. The student was presented an increment of information then tested on the concept involved. If the student responded incorrectly, he was given a simplified and expanded version of the same information and allowed to proceed. More advanced programmed instruction methods soon developed with a branching structure capable of allowing brighter students to progress at a faster rate and tailoring the remedial information to the mistaken response given.

Computer-assisted instruction then implies the implementation of programmed instructional concepts on a digital computer. The interaction, feedback, and reinforcement specified above make an interactive graphical approach especially well suited to computer-assisted instruction.

5. Basic Logical Design

By far, the oldest area of interest to the proposed project was that of logical design. In 1854 George Boole, an English mathematician, published his classic book: An Investigation of the Laws of Thought on Which Are Founded the Mathematical Theory of Logic and Probabilities. Proceeding from his basic investigation of classical logic, Boole derived a "logical algebra" which today bears his name.

The ability of boolean algebra to adequately describe the behavior of relay switching circuits was first recognized by C. W. Shannon, also of MIT. In his Masters Thesis: "A Symbolic Analysis of Relay and Switching Circuits," [ref. 14] Shannon showed that any circuit consisting of combinations of switches and relays could be represented by a set of mathematical expressions. He further showed that these expressions were exactly equivalent to the algebra derived by Boole in the field of symbolic logic. Thus boolean algebra finds much application in the design of digital computer systems composed of storage elements and their associated circuitry for switching from one state to another.

Boolean algebra differs from ordinary algebra in some fundamental ways. As in ordinary algebra, letters are used as terms in boolean expressions but their meaning is different. Boolean variables can take on only two distinct values (usually represented by the binary numbers 0 and 1). Thus boolean variables are useful for depicting the existence

or non-existence of a given condition, such as a switch being open or closed or a statement being true or false. Boolean functions may be formed by using a number of different operators. However, all of these operators are derivable from sets of primitive boolean operators. One commonly known set consists of intersection (AND), union (OR), and negation (NOT).

At the basic level taught in an introductory course, logical design involves the use of boolean primitives to describe information flow in a digital computer. Additionally, since a digital computer is a finite state machine, information storage requirements must be considered. Temporary storage (registers) or permanent storage (core) is accomplished in a computer by some type of bi-stable device. For design applications, register storage is usually of most interest and is accomplished by means of a flip-flop (bi-stable multivibrator).

Implicit in the assumption of a binary storage element is the ability for this element to be able to change state. Hence the value of Shannon's thesis is the ability to describe the conditions necessary for the switching of element states in boolean terms. The types of flip-flops thought to be of most application in a basic logical design course were the clear-set flip-flop, clear-set-trigger flip-flop and J-K flip-flop. State diagrams for these various flip-flops as well as truth-table representations of the boolean primitives AND and OR are found in the Appendix.

III. PROGRAM IMPLEMENTATION

A. COMPUTER SELECTION

With the above background information reviewed and under the assumption that an interactive learning aid for the design of basic computer circuits would be of value in the instruction of Naval Postgraduate School students, the actual construction of the program began. Of primary concern was the selection of the computer installation upon which to implement the program. Prior statements emphasizing the need for interaction and graphical display inherent in the problem narrowed the choice to the Xerox Data Systems 9300/Adage Graphics Terminal - 10 (AGT-10) system in the Naval Postgraduate School Computer Laboratory. It was felt that in achieving "minimal interference with other computer users", the logic design program would be best implemented on the AGT-10 system using this system's "stand alone" capability.

The main consideration in the selection of this computer facility was the significant ability of the Adage Graphics terminal to be programmed for an interactive instructional application. The graphical display capability of this equipment, coupled with the interfaced user communication devices provided an ideal research vehicle for this computer graphics task. However, this is not proposed as a cost-effective way to build a learning aid.

A large, fast processor (XDS 9300) capable of accepting higher level language programs, tied to a separate smaller computer (AGT-10) responsible for the graphical display was a concept used in many of the circuit design implementations cited above. However, in view of the

program's proposed use as a learning aid, it was felt the increased availability of the program to potential users offered by "stand alone" implementation would be of value in earning student acceptance of the program.

The "stand alone" capability of the AGT-10 is available by virtue of the fact that the Adage graphics unit has its own processor. The graphics facility provided by the AGT-10 system is comprised of a high speed, high precision vector generator with cathode ray tube (CRT) display [ref. 15]. The CRT has a 12 by 12 inch display area, with a smaller area of high resolution. A stroke-type character generator is also available for CRT display of alphanumerics. The graphics console of the AGT system has additional devices incorporated which facilitate graphical communication with the user. Devices of special interest to this program were the function switch box, light pen and alphanumeric keyboard.

The DPR-2 Digital PRocessor associated with the Adage Graphics Terminal system is a general purpose digital computer with extensive transfer logic and addressing capabilities. The processor has a 30 bit word length, memory cycle time of two microseconds and 8 K memory size. Additional random-access memory is provided by the DMS-2 Disk Memory Subsystem.

The main software support for the AGT-10 system is the Adage Extendable Program Translator (ADEPT). ADEPT is an open-ended string substitution macro translator capable of producing relocatable machine language code. Two passes of the source language (ATEXT) are made to allow unlimited forward references to symbolic addresses. Other features of the ADEPT translator are:

- 1) Automatic definition of location symbols.
- 2) Parameter assignment statements.
- 3) Macro nesting capability.
- 4) Conditional translator capability.
- 5) Definite and indefinite repeat statements.

A unique feature of ADEPT allows definition of "action operators" in addition to those already present in the ADEPT translator; thus allowing the programmer to extend the language to his own needs.

B. FLOW - OF - CONTROL

Preliminary to the actual coding of the program in the ADEPT language, a consideration of overall program flow of control and data structure was undertaken. It was decided to divide the program into three main modes (as in CIRCAL). These were DEFINE, CONNECT, and ANALYZE. Student specification of the circuit, selection of analysis, and observation of circuit response to specified input values would be accomplished within this main framework. In addition, a brief instructional mode would be provided within which the student would be shown necessary information to operate the program. A more expanded instructional mode would display various basic logic circuits with inputs specified and outputs displayed with instructive comments. Program flow of control would allow the student to specify various inputs to a particular circuit and observe its response, change the circuit structure by addition or deletion of circuit elements, or change the interconnection of these elements. The instructional modes would be capable of selection from any point in the program and were intended to be comprehensive enough to allow the student to operate the program without any additional instruction.

C. DATA STRUCTURE

Of utmost concern at the outset was the selection of a data structure which was sufficiently powerful to represent the display structure and topological structure of a logic circuit, yet was capable of straightforward implementation at the relatively low programming level of the ADEPT language. In most of the circuit analysis programs reviewed, the data structure used was some type of ring structure, involving a complex system of pointers. The complete ring structure requires having pointers from each data item to the preceeding and following item. Unlike a list, the structure is closed by also having pointers between the first and last data items. The inherent advantages of this type of structure are ease in searching for data items and the ability to represent multidimensional concepts wherein one data item is a member of more than one ring. Connectivity relations, set membership, and terminal node identifications are especially well represented in this manner. This concept was implemented by Sutherland in his "generic" data structure which grouped elements of the drawing by common properties [6].

While the ring data structure has these circuits representation benefits, it also creates implementation problems in an assembly level language such as ADEPT. In order to handle the more sophisticated structure, many primitive operations on the structure itself must be programmed at the machine level whereas this is not a requirement if a higher level language is used [ref. 16].

Much recent work in the area of computer data structures has preceeded from the basic foundations of graph theory. Robin Williams [ref. 17] has shown the advantages of the graph theory approach to preserve the relationships and logical associations that exist among data

items in a computer program. One such approach involves the use of a bipartite graph. The bipartite graph consisting of two distinct types of vertices (in this case inputs and outputs) is especially well suited for representation of logic circuits. This graph has been defined [ref. 18] as one whose vertices can be partitioned into two disjoint sets in such a way that every edge has its first end point in one set of vertices and the other end point in the remaining vertex set.

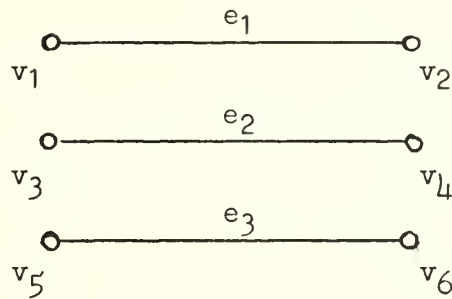


FIGURE 1

A Bipartite Graph

The circuit representation advantage offered by this approach to the data structure is that each gate may be considered as the intersection of two distinct types of pointers. Thus, the circuit is completely represented at each element by a set of pointers for inputs and one for outputs.

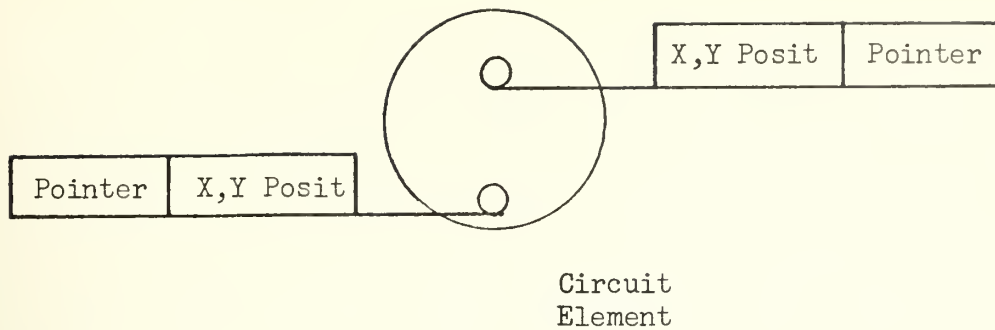


FIGURE 2

Graph Theory and Data Structure Representation of an Arbitrary Circuit Element

Another benefit of this method of representing circuits is that it does not pose a limit on the number of inputs or outputs which may be connected to any specific gate.

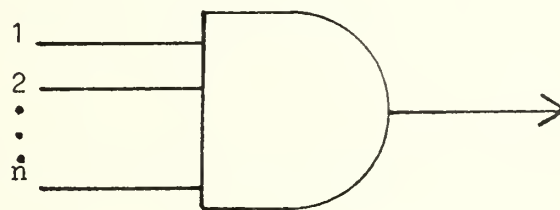


FIGURE 3

AND Gate with n Inputs

It was decided to use the bipartite graph method of circuit representation in the computer program. However, the problem of depicting this structure at the ADEPT level remained. Thus an older means of representing data structure was implemented involving establishment of table structures in contiguous areas of memory. Since the number of circuit nodes the program was designed to handle was limited, these tables could be made a fixed length. Pointer structure was implemented

at the table level in that the head item in a data table pointed to the last item and vice versa, but internal pointers within each table were not present. These were represented by the contiguity of the table. The pointers representing the edges of the bipartite graph were placed in these tables as were the X,Y coordinates representing circuit element location.

This model of the topology of the circuit consisting of direct access data sets in fixed length blocks has several benefits. It is extremely economic in its use of storage, data access time is short and the capacity of the tables can be easily increased. An additional benefit brought to light by this implementation is that display tables can be "preloaded" with display control instructions. This is especially important in the display of character information for establishing size, brightness, and italics control information.

D. MAIN PROGRAM ROUTINES

The INTERACTIVE LOGIC LABORATORY is composed of five distinct ADEPT programs designed to accomplish the objectives discussed above. The separate relocatable versions of these programs are linked together at execution time along with system routines FIN (for checking function switches) and AMRMX (for teletype interface). The five programs are hierarchically related as follows: The main program called LIL is responsible for displaying all of the user-entered and program-generated information. LIL calls four subordinate routines: LOGMM, CONCT, ANALR and INTRO based on the output of the user's light pen. LIL enables this instrument for "hits" on the text words DEFINE, CONNECT, ANALYZE and INTRODUCTION and branches to the appropriate subprogram. Thus, LIL is also responsible for the overall flow of control as specified by the user.

1. Define Mode

The ADEPT program LOGMM accomplishes the circuit element definition objective. LOGMM creates the display of the logic circuit by allowing the user to draw AND gates, OR gates, inverters and flip-flops on a 5 by 5 grid displayed on the CRT. (Only the intersections of the grid which represent possible circuit node locations are drawn as dots). A "menu" of available circuit elements is displayed at the bottom of the presentation and each menu item is labeled with the appropriate function switch which will cause a copy of that element to be drawn.

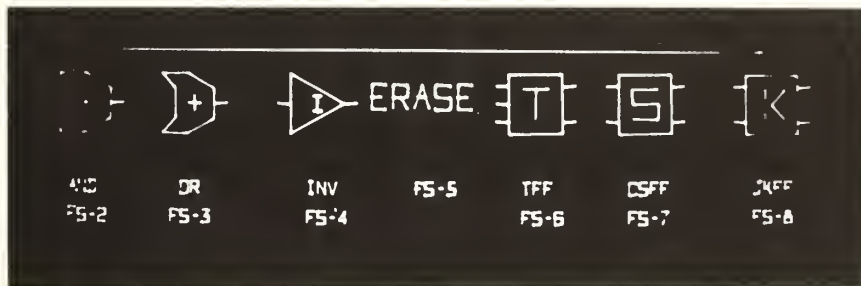


FIGURE 4

"Menu" Display

Actual circuit construction is accomplished by the user with the light pen. This instrument is used to select the dot where the circuit element is desired. Selection is indicated by the appearance of a square cursor around the dot. Depressing one of the appropriately labeled function switches chosen from the menu causes an instance of that circuit element to be entered into the programs display table called TBL0.

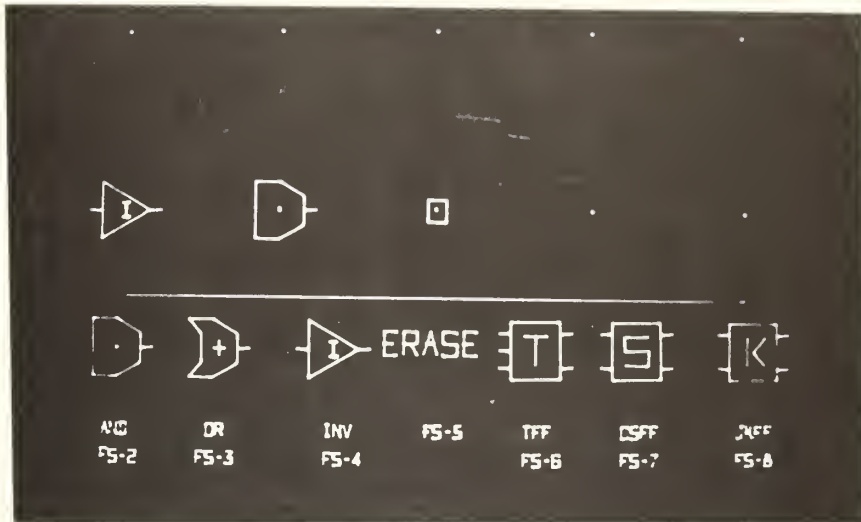


FIGURE 5

Entering Circuit Elements in DEFINE Mode

X and Y display coordinates are entered into the display table along with types of gate in the following format:

bit		131415		2627 29	
0					
X display coordinate (15 bits)		E	Y display coordinate (12 bits)		type (3 bits)
		O			
		L			

FIGURE 6

TBLO Table Entry

The top entry in the TBLO Table is a pointer to the last entry in the table. The end-of-list bit (abbreviated EOL above) is not used. The gate types are entered according to the following table:

000	--not used
001	--AND gate
010	--OR gate
011	--inverter
100	--clear-set flip-flop
101	--toggle flip-flop
110	--J-K flip flop
111	--not used

TABLE 1

Gate Types Entered in TBLO

Removal of a circuit element is accomplished by selecting an existing element with the light pen (a similar cursor indication notifies user of selection). This is accomplished by removing the erased entry from the TBLO table and moving all subsequent TBLO entries up one location. The pointer to the last entry is adjusted to point to the new last entry in the table. Upon exit the program returns to the main display mode.

2. Connect Mode

The program CONCT establishes the topological structure of the circuit. Its major function is to build the display table for the connecting lines between circuit elements previously entered in the DEFINE mode. Function switches are implemented in this mode for moving the cursor and for drawing lines to represent the circuit connections. The created lines are entered into a display data set called DATA1 and refreshed continually. Additional function switches allow the user to select any point on the screen to which he has previously moved the cursor. The points may be selected in the ordered entered (select forward) or in reverse order (select backward). This feature aids the user in "hooking" a line for subsequent erasure from the data set or in positioning the cursor exactly on a point previously moved to or drawn.

Terminal connections of the circuit are drawn in the usual schematic manner. Data entered in DATA1 table is of the same format as the normal 30-bit Adage display word.

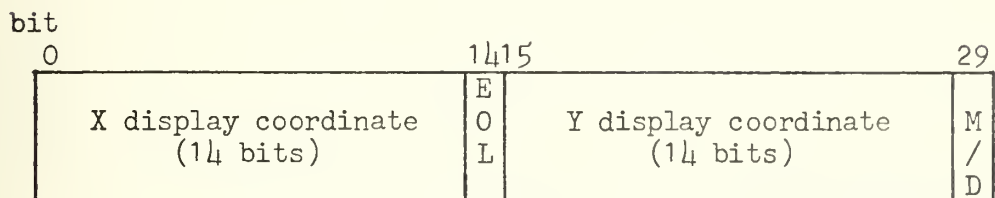


FIGURE 7

DATA1 Table Entry

The end-of-list bit (EOL) is present to one in all table entries. The remaining bits in the DATA1 table are zeroed. When the user enters a "move" or "draw" command via the function switches in the CONNECT mode the display coordinates of the cursor are stored in the next DATA1 entry. The EOL bit is cleared and the move-draw bit (abbreviated M/D above) is set if the command was a "draw". When this information is displayed, the presence of an end-of-list bit following the last DATA1 entry is assured. The DATA1 data set thus is drawn as one contiguous set of display commands.

3. Analysis Mode

The program ANALR performs analysis of the drawn circuit in two ways. Upon entering ANALR for the first time, the display tables of circuit elements created in the DEFINE mode and circuit connections entered in CONNECT are examined to determine the topology of the network.

This is accomplished by creating a bipartite graph with circuit element inputs and outputs considered as the vertices of the graph and pointers representing edges. The results of this analysis are stored in

other memory tables to be used subsequently in calculating circuit response to specified inputs.

a. Circuit Analysis

The topological analysis rests on this assumption: Any circuit element (or terminal node) which is connected to another element will have a line (or lines) representing this connection in the data set created in CONCT.

Thus, the first analysis task is to determine the end points of the connections entered. This is done by searching the DATA1 table for an instance of a "move" entry. The assumption is made that this "move" entry indicates the user is about to draw a sequence of lines representing a circuit connection. The significance of the "move" entry is that the X,Y position of the "move" will be the location of the initial point of the connection. Hence the X,Y position identified as a "move" is placed into a table called RAWBK. The succeeding DATA1 entries are searched until the next "move" entry is found indicating a new connection sequence has begun. The X,Y position of the immediately preceeding entry in the table is then recorded as the termination of the connection established by the original "move". This X,Y position is then entered into the next position in the RAWBK table. The above process continues until the entire DATA1 table has been searched. Special care is exercised to ensure that extraneous moves inadvertently entered by the user, or erased entries do not affect the extraction of topologically correct line end points.

DATA1	bit	bit	RAWBK	bit	bit
0	14	29	0	14	29

display X1	0	display Y1	0
" X2	0	" Y2	1
" X3	0	" Y3	1
display X4	0	display Y4	1
display X5	0	display Y5	0
. .	0	. .	1
. .	0	. .	1
. .	0	. .	1
display Xn	0	display Yn	1
0 0 0 0		0 0 0 0	
0 0 0 0		0 0 0 0	

display X1	0	display Y1	0
display X4	0	display Y4	1
display X5	0	display Y5	0
display Xn	0	display Yn	1
0 0 0 0		0 0 0 0	

FIGURE 8

Relationship of DATA1 and RAWBK Data Tables

The format of the entries in the RAWBK table is exactly the same as that of DATA1. The difference is that the RAWBK table contains only those entries in DATA1 which represent the end points of connections. Thus the RAWBK table is a compression of the topological information contained in the DATA1 display table.

Once the RAWBK table has been built, the basic information required for topological analysis of the circuit is present. The actual construction of the bipartite graph begins by establishing a circuit scanning loop. In this loop, each column of the grid is searched proceeding from the left to the right side of the CRT. This is done assigning an X coordinate value to the variable "level" which will be greater than any X value of a circuit termination point connected to a circuit element in that column of the grid. However, the "level" value is less than the X value of an entry in the next grid column.

All termination points entered in RAWBK which have X values to the left of this level will be mapped into one of three tables. If the display coordinates of a termination point fall within the limits

which allow a point to be connected to a gate, a determination is made whether this point should be considered as an input or output to this element. This decision is based on the side of the gate to which the point is connected. If a RAWBK entry falls within the X,Y constraints and is an input, a corresponding entry is made in the BAND (Basic ANalysis Data) table. Outputs are similarly entered into the BANDO (Basic ANalysis Data, Outputs).

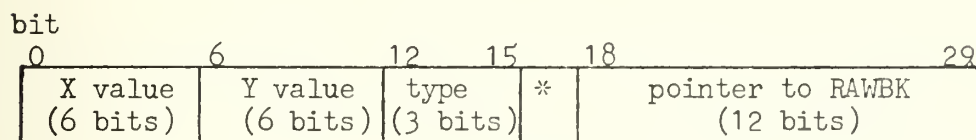


FIGURE 9

BAND Table Entry

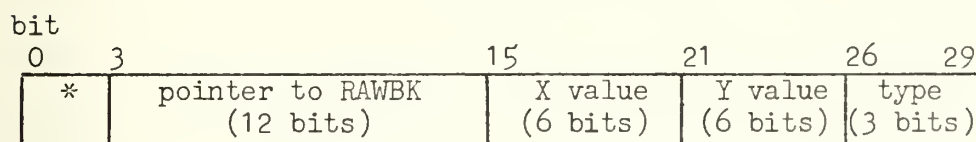


FIGURE 10

BANDO Table Entry

The field labeled * above represents a three bit value corresponding to the height of this input (or output) on the gate. This information must be extracted for use in the analysis of sequential elements.

Any RAWBK entry whose X value is to the left of the level value but does not map into a circuit element is placed in the unresolved (URD) table. (Note that all circuit inputs and outputs will generate an URD entry as one of the end points of each of these lines is not connected to a gate). The form of the unresolved data (URD) table is the same as that of RAWBK: An X,Y display coordinate value. The coordinates entered into Inputs/Outputs table (BAND/BANDO) are compressed to allow room in

the word for the gate type and connection pointer determined by this phase of ANALR. The X,Y coordinate values entered are the two most significant (Octal) digits of the display coordinate of the circuit element entries in the TBLO table. This was done because the X,Y values of the defined entries in TBLO are fixed by grid position whereas the RAWBK X (and Y) entries can vary significantly depending on the width (or height) of the circuit element. Four digits representing X,Y position together with one digit representing the type of gate are thus extracted from the TBLO entry and packed into one half word of the Inputs/Outputs table. The pointer half word contains the RAWBK address of the other end point of this connection. To make explicit the bipartite graph implication of the pointer, a BAND entry has its pointer in the lower half word (telling where this input goes) and BANDO entry pointers are in the upper half word (telling where this output comes from).

In addition to the URD (Unresolved) table, which contains the display coordinates of inputs and outputs in a full 30 bit word, the unresolved entries in RAWBK are used to create another table called "Un-Resolved Inputs/outputs" (URI). Each entry in the URI table consists of two pointers. The pointer in the top half word designates the display coordinates of this unresolved entry (i.e. a pointer into the URD table). The lower half word is a pointer to the memory location in the ANSW (answers) table reserved for the actual binary value of the input (or output) which will be specified (or calculated) later in the program. This table is required to enable user specified inputs to be placed into the proper ANSW (answer) location. Complete topological analysis is thus attained by repeating the above procedure for each grid column.

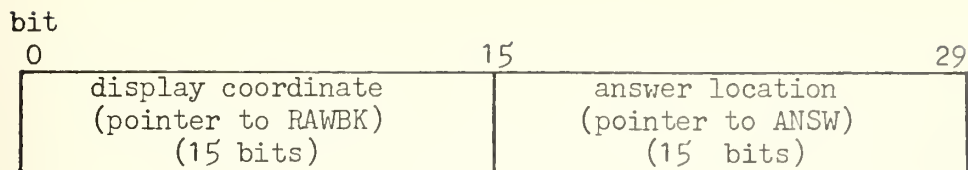


FIGURE 11

URI Table Entry

b. User Input

The entire connection analysis outlined above is transparent to the user. After its completion, ANALR (Analysis Program) automatically enters a specification phase. Two additional kinds of information are needed before circuit response can be calculated. The first of these is the labeling of terminal nodes of the circuit with up to 3 alphanumeric characters to enable a logic equation representing the circuit to be constructed. Once nodes have been labeled, actual binary inputs are accepted for each input preparatory to calculating circuit response. The node labeling phase is especially important because this is where the user will be notified of an improper circuit connection if one exists. Any entry (not a circuit input or output) which was not properly connected by CONCT will be pointed out for labeling. Hence, if the program asks for a label where the user can see that one is not required, the circuit must be re-connected.

The user is allowed to label the circuit with up to three alphanumeric characters at each circuit input and output. These characters are accepted and processed by ANALR (Analysis Program) and converted into display form for subsequent drawing on the cathode ray tube by LIL (main display mode). In order to do this, a foreground/background type of operation is set up between LIL and ANALR which allows

the continual display of already entered circuit information while waiting for teletype inputs of alphanumerics for subsequent display. The binary values input to the circuit are then accepted and processed in a similar manner. All character display information is stored in one of two tables: CTAB for label display data and NTAB for value display data. The structure of each of the tables is exactly the same.

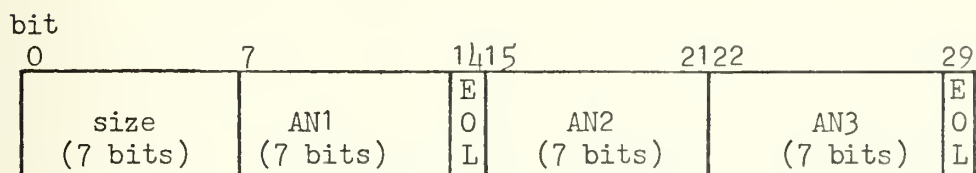


FIGURE 12
CTAB/NTAB Table Entry

The abbreviation AN above is for Alphanumeric character field, capable of storing one ASCII (American Standard Code for Information Interchange) code display character.

c. Circuit Response

The response of the circuit to specified inputs is then determined. Another table called ANSW is reserved in memory for holding the binary numbers input, and output values at any point in the circuit. Data representing both is indexed into this table by the last two digits of the RAWBK pointer in the corresponding BAND or BANDO entry, thus guaranteeing a unique storage address for each input and output value.



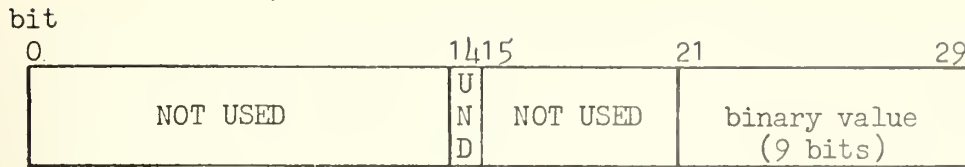


FIGURE 13
ANSW Table Entry

Another table reserved in memory is FFDT (Flip-Flop Data Table). This table is actually built upon exit from the DEFINE mode. One entry is made in this table for each flip-flop entered in DEFINE. The format of the entries is the same as the standard Adage display word: A 14 bit display X value, a 14 bit display Y value and end-of-list bits in bits 14 and 29 of the display word. The EOL bit in bit position 14 is used as an undefined flag (as in ANSW above). The EOL at bit position 29 contains the current state of the flip-flop: (set-1, reset-0). The undefined flag will be set when the state of the flip-flop cannot be calculated from current inputs.

Actual response calculation is done using a memory stack. The STACK is loaded from the top down with the binary values from ANSW which represent the inputs to the gate being simulated. A subroutine call is made to one of six subroutines to simulate the response of the circuit element to these inputs. The subroutine call is based on the gate type [Table 1].

The subroutine operates on all the stack values and places the binary result on top of the stack. Upon return to the main analysis program, this value is stored into the appropriate memory location in ANSW (answers). If the circuit element being simulated is a flip-flop some additional work must be done. Before the call to the appropriate

simulation subroutine the last state of the flip-flop must be obtained from the FFDT (Flip-Flop Data Table). This is stored at the location STACK-1 and is used to calculate the response of the flip-flop to the current input based on the last state. Upon return from the subroutine the current state of the flip-flop is entered into the appropriate FFDT location.

Circuit response calculation is carried out from left to right according to the following algorithm.

- 1) Locate the first (or next) circuit element in the BAND table.
- 2) Find the binary value in ANSW (specified by this BAND entry) and load this value into the stack.
- 3) Find all other inputs to this gate in BAND and load their values into the stack.
- 4) Calculate response of this gate by a jump to the appropriate subroutine as specified by type of gate.
- 5) Locate each instance of this gate in the output (BANDO) table and stuff the calculated value into the ANSW address specified by this entry.
- 6) If this is the last circuit element stop, if not go to step 1.

Once the response of the complete circuit has been calculated, the answers must be displayed to the user. This is done by converting the binary value in each location in ANSW corresponding to a circuit output into display form and entering it into the number display table.

After displaying the results of the calculation ANALR returns to the "INPUT VALUES" mode to allow the circuit to be tested with other input values. A new circuit may be constructed (or the current one modified) by leaving the analysis mode via the "MODE EXIT" function switch.

IV. PROGRAM EVALUATION

A. SAMPLE TERMINAL SESSION

The following computer session is documented as an example of the abilities of the program to design logic circuits. The objective of the session was to design a half-adder according to the logic expression $\bar{A}.B + A.\bar{B}$, the exclusive-or representation of a half-adder circuit. All of the following photographs were taken from the CRT. Upon program initiation the main display mode was entered resulting in the following display:



FIGURE 14

Initial Command Presentation

Selection of INTRODUCTION would display to the user a comprehensive explanation of the program and how to use it.

In this example case the DEFINE mode was selected with the light pen and the basic circuit elements of the half-adder were entered. It was desired to design the half-adder according to the above mentioned exclusive-or representation, hence two inverters, two AND gates and an OR gate were entered as shown by the following display in the DEFINE mode:



FIGURE 15

DEFINE Mode Display

After exiting the DEFINE mode (which created the above display) via the function switches the CONNECT mode was selected from the command menu of the main display mode. Connections were then established by moving the cursor with the appropriate function switches. A typical CONNECT view is shown in the following scope photograph:

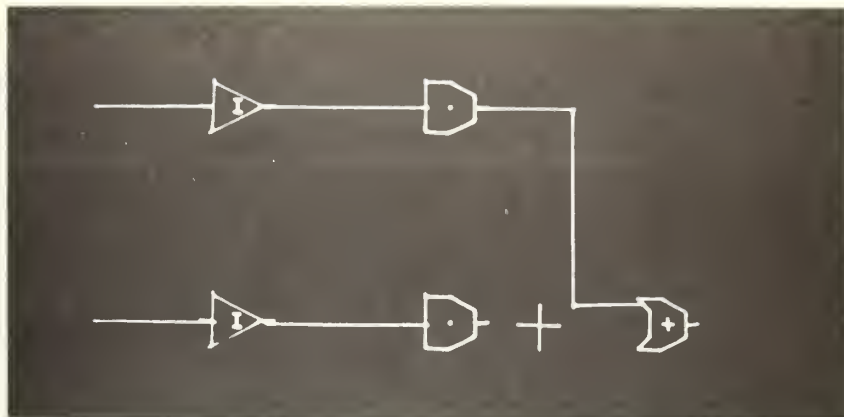


FIGURE 16

CONNECT Mode Display

The circuit was then completely connected and ANALYZE was selected after exit from the CONNECT mode to the main display mode. Initial selection of this mode causes the program to extract the topological information contained in the circuit and build the appropriate tables as discussed above. The following display signifies to the user that the program is ready for labeling of the circuit terminals and the subsequent specification of binary input values:

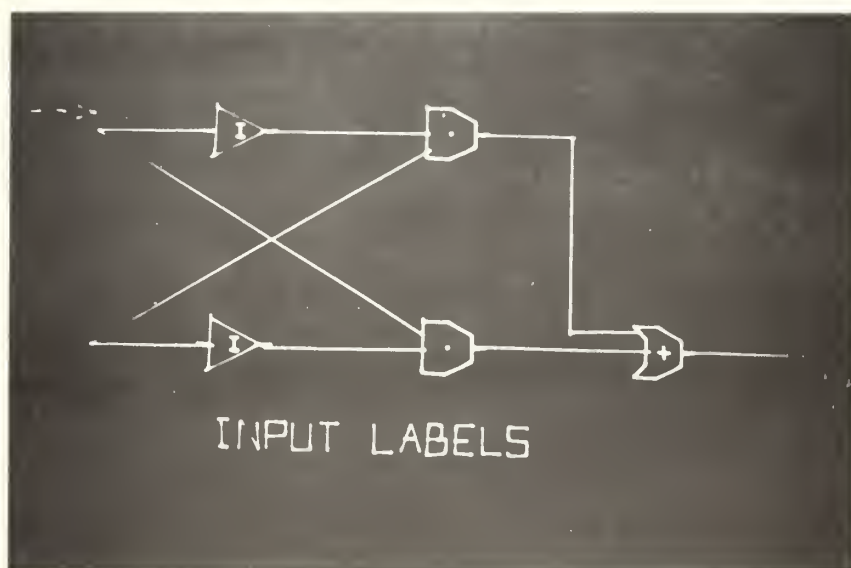


FIGURE 17

ANALYSIS Mode--Alphanumeric Input Phase

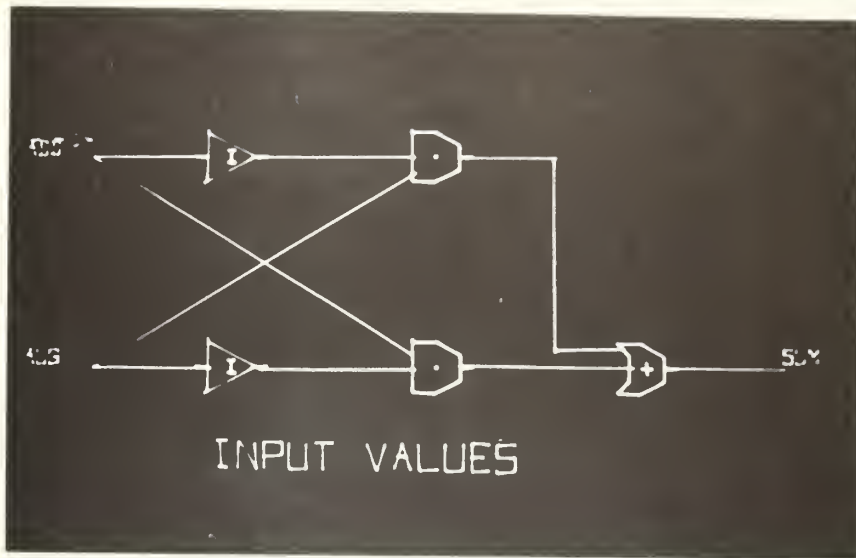


FIGURE 18

ANALYSIS Mode--Numeric Input Phase

As previously mentioned any circuit element whose connection is not properly specified upon initial entry into this phase of the program will be pointed out for labeling. When the labeling process is completed the program automatically enters the response calculation mode, wherein the user's binary teletype inputs are accepted and processed for each circuit input. The appropriate routines are called to simulate the circuit and the result of the response calculation is converted into display form and is shown for this particular circuit in the following picture:

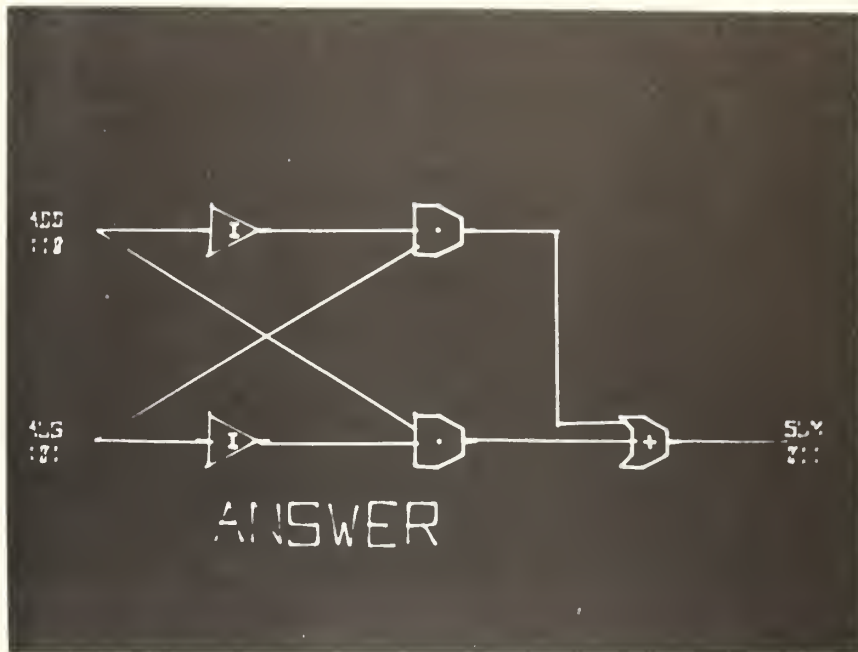


FIGURE 19

ANALYSIS Mode--Display of Response

At this point the user can go back to the main program via "mode exit" function switch or can retest the circuit with new binary inputs by depressing function switch five ("reset"). This function switch will cause the program to return to the "INPUT VALUES" display shown previously.

The following sequence of CRT photographs shows the response of a set-toggle-reset flip-flop to various binary inputs. The flip-flop is reset initially as indicated by the zero value displayed at its output terminals.

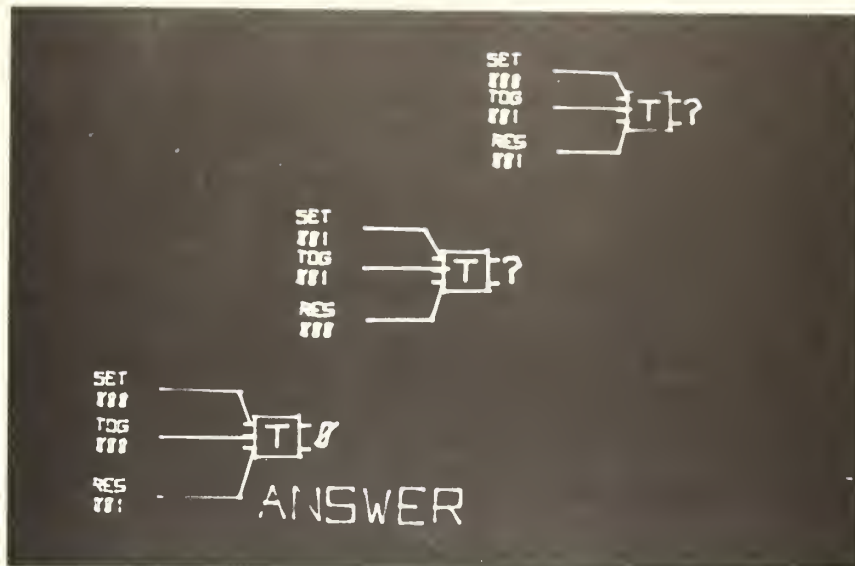


FIGURE 20

Flip-Flop Analysis(1)

A pulse applied to the set terminal causes the flip-flop to change to the "set" state (Lower left flip-flop).

Two subsequent toggle inputs change the flip-flop's value appropriately (Middle and upper right flip-flops).

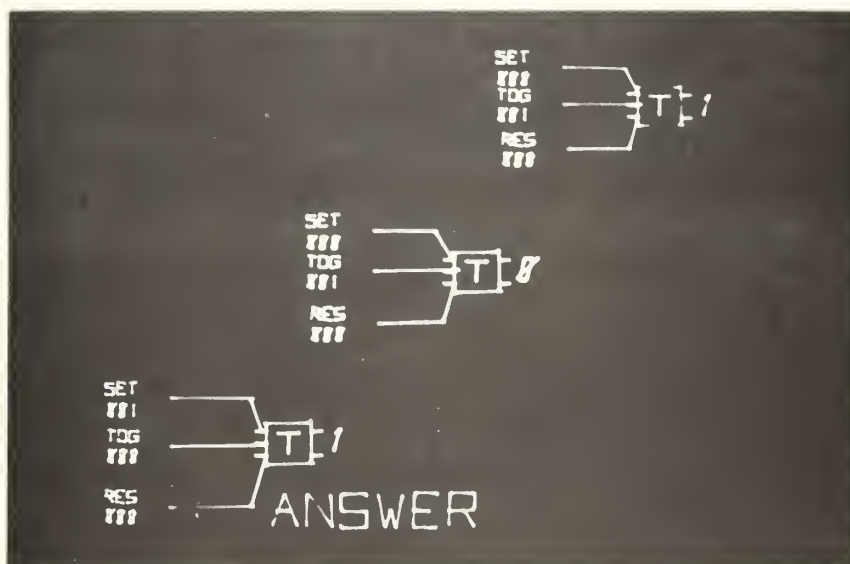


FIGURE 21

Flip-Flop Analysis(2)

A reset pulse clears the flip-flop to zero (Lower left flip-flop).

Finally the flip-flop's response to an illegal combination of inputs is shown.

B. RESULTS

At the current stage of program development the INTERACTIVE LOGIC LABORATORY has demonstrated significant ability to enable the user to create and analyze basic logic circuits. The program has not yet been used by beginning students in logical design. However, several features of the implementation should allow its eventual use as an adjunct to the classroom instruction received in the "Logical Design of Digital Computers" course (CS-3200) taught at the Naval Postgraduate School.

- 1) Although no generally accepted set of symbols exists for representing logic elements, the symbols used are universally identifiable since the specified operation is shown in the logic display symbol.
- 2) The sequential elements (flip-flops) are not standard symbols, but are quite recognizably presented.
- 3) The degree of student control achieved by the implementation allows the student to proceed at his own rate in the design of basic logic circuits.
- 4) The level of description of the INTRODUCTION mode is sufficiently comprehensive to allow most students to use the program with no assistance. Additionally, the fact that this mode does not have to be selected and only part of the instructions may be reviewed does not subject the student to tedious repetition of instructions as he gains proficiency in the use of the program.



Implementation of sequential circuits has met with less success. At the current stage of development the INTERACTIVE LOGIC LABORATORY can not be used for the design of sequential circuits. However, individual flip-flops are simulated correctly thus allowing the student to observe the response of individual elements to various inputs.

C. EXTENSIONS

Two basic inadequacies exist in the program. First of all, the bipartite graph representation does not allow any feedback loops to be present in the circuit. Since this is not an uncommon occurrence in logic circuits, the program should be modified to allow the output from a circuit element to be delayed and re-input to the same element. Secondly, the program as currently implemented does not allow for recursive constructs, hence only circuits that will fit on the 5 by 5 grid may be constructed. It is felt that the bipartite data structure is general enough to handle larger circuits, wherein a previously analyzed circuit is reduced to be considered as another primitive element.

Another area where extension of the current program is required is in the saving of a user's circuit design efforts and hard copy output. The saving of circuits could be done by punching out the display tables on paper tape. Then at a subsequent computer session the DATA1 and TBLO tables could be read into the teletype unit allowing the user to continue where he previously stopped. Hard copy output which is intelligible to the user is harder to obtain with the AGT-10, but the cheaper graphics terminals required for more general implementation of this program have provisions for hard copy output.

D. RECOMMENDATIONS AND CONCLUSIONS

It is felt that INTERACTIVE LOGIC LABORATORY has shown the feasibility of using a computer graphics terminal to demonstrate the basic concepts of logical design. One of the severe drawbacks associated with this particular implementation is the high cost of the Adage Graphics Terminal. While this could be justified by the research intent of this program, any extension of the concepts of this program into general classroom use will necessitate a lower cost graphics terminal.

Currently available storage tube graphics terminals meet these cost requirements. A classroom installation composed of individual cathode ray tubes for each student tied to a central computer capable of responding to all users in a time-shared mode would be ideal for the implementation of this logic demonstrator. Such a classroom computer installation is now available and the implementation of the INTERACTIVE LOGIC LABORATORY on this equipment would indeed provide a meaningful laboratory for experimentation in basic logical design.

APPENDIX A

Truth Table for 3-input gates

			AND	OR
A	B	C	$A \cdot B \cdot C$	$A + B + C$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Truth Table for clear-set flip-flop

SET	RESET	INITIAL STATE	FINAL STATE
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	? (undefined)
1	1	1	? (undefined)

APPENDIX A (continued)

Truth Table for clear-set-toggle flip-flop

SET	TOGGLE	RESET	INITIAL STATE	FINAL STATE
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	?
0	1	1	1	?
1	0	0	0	1
1	0	0	1	1
1	0	1	0	?
1	0	1	1	?
1	1	0	0	?
1	1	0	1	?
1	1	1	0	?
1	1	1	1	?

APPENDIX A (continued)

Truth Table of J-K flip-flop

J	K	INITIAL STATE	FINAL STATE
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

JPAR	DD	[JUMPS TO DOT DRAW IF TABLE EMPTY
MDAR'X	GCNT	
MDAR'F	TBL9	
MDAE	GCNT	[GET NEXT ADDRESS IN TABLE
ARM0	T1	
MDAR'I	T1	[LOAD CONTENTS OF TBL9 PLUS GCNT
MDAR'A	C3	[MASK ALL BUT ID BITS
MDAE'N	C1	[PICK PROPER SUBROUTINE
JPAR	ADDP	
MDAE'N	C1	
JPAR	GRP	
MDAE'N	C1	
JPAR	INVP	
MDAE'N	C1	
JPAR	FFT1	
MDAE'N	C1	
JPAR	FFS2	
MDAE'N	C1	
JPAR	FFK3	
MDAR'X	FLG1	[SET ADD FLAG
MDAR'I	T1	[LOAD DX AND DY INTO AR
MDAR'A	C2	[MASK ID BITS
ARM0	DADD	
JPSR	DRW	
ARX0'F	FLG1	[RESET FLG1
ARM0	RETUR+2	
JUMP	FLG1	[SET FLG1 TO 2
MDAR'X	C1	
MDAE	FLG1	
ARM0	T1	
MDAR'I	C2	
MDAR'A	D0RD	
ARM0	DRW	[OR DATA
JPSR		
ARX0'F		
ARM0	FLG1	

ADDP:

GRP:

INVP:	JUMP	RETUR+2	
	MDAR'X	FLG1	
	MDAE	C4	[SET FLG1 T9 3
	ARM	FLG1	
	MDAR'I	T1	
	MDAR'A	C2	[INV DATA
	ARM	DINVD	
	JPSR	DRW	
	ARX0'F		
	ARM	FLG1	
	JUMP	RETUR+2	
FFT1:	MDAR'X	FLG1	
	MDAE	C6	[SET FLG1 F9R FFT
	ARM	FLG1	
	MDAR'I	T1	
	MDAR'A	C2	
	ARM	FFTD	[FFT DATA
	JPSR	DRW	
	ARX0'F		
	ARM	FLG1	
	JUMP	RETUR+2	
FFS2:	MDAR'X	FLG1	
	MDAE	C6	
	MDAE	C1	
	ARM	FLG1	[SET FLG1 F9R FFS
	MDAR'I	T1	
	MDAR'A	C2	
	ARM	FFSD	[FFS DATA
	JPSR	DRW	
	ARX0'F		
	ARM	FLG1	
	JUMP	RETUR+2	
FFK3:	MDAR'X	FLG1	
	MDAE	C4	
	MDAE	C6	
	ARM	FLG1	[SET FLG1 F9R FFK

T1	FLG1	GCNT	D3	DINVD
C2	RETUR+2	DTDRW	DRW	DRW
FFKD		D1	LFLG	LFLG
DRW		DADD	+2	+2
		FLG1	-2	-2
		DRW	FLG1	FLG1
		LFLG	D2	D3
		+2	DORD	DINVD
		-2	DRW	DRW
		FLG1	LFLG	LFLG
		D2	+2	+2
		DORD	-2	-2
		DRW	FLG1	FLG1
		LFLG	D3	D3
		+2	DINVD	DINVD
		-2	DRW	DRW
		FLG1	LFLG	LFLG
		D4	+2	+2
		FFTD	-2	-2
		DRW	FLG1	FLG1
		LFLG	D4	D4
			FFTD	FFTD
			DRW	DRW
			LFLG	LFLG

● ●
Σ
D

THE
DANCE.

[CONTINUE MENU DRAW

JPLS
JUMP
MDAR'X
MDAR
ARM
JPSR
MDAR
JPLS
JUMP
MDAR'X
MDAR
ARM
JPSR
MDAR
JPLS
JUMP
ARX0'F
ARM
ARX0'F
ARM
MD10'A'L;
MDIC'A'L;
MDAR'F
ARM
MDAR'F
ARM
MDIC'0'L;
MDAR
JPLS
JUMP
MDIC'A'L;
MDAR
JPAN
JUMP
JPSR
MD10'0'L;
•+2
•-2
FLG1
D5
FFSD
DRW
LFLG
•+2
•-2
FLG1
D6
FFKD
DRW
LFLG
•+2
•-2
FLG1
LFLG
17777JH
77763
TXT-1
77735
TE9LR
77736
10
LFLG
•+2
•-2
77763
XFLG
•+2
•+2
INDRW
1000JH

TXTW:

[TURN 0FF VECTOR GEN
[SET IC BITS 26,27

[TURN 0N CHAR GEN

[RESET IC BITS 26,27

INDRA:

MDAR	SAVDR
JUMP'I	RETUR
N90P	
ARMD	SAVTX
MDIC'A'L'	-10
MDIO'0'L'	60000JH
MDAR'X	LFLG
MDAR	SAVTX
JUMP'I	TE9LR
N90P	
ARMD	SAV9
MDIO'A'L'	-20JH
MDAR'I	77756
MDAR'A	C1
JPLS	•+7
MDAR	77756
MDAE'N	C1
ARMD	C5
MDAR'I	C5
ARMD	DXDY
JUMP	•+3
MDAR'I	77756
ARMD	DXDY
MDAR'H	SCALD
MPYU	DXDY
N90P	
MDAR'A	AA1
ARMD	TEM
MDAR'H	SCALD
MPYL	DXDY
N90P	
ARRS	17
N90P	
MDAR'A	A2
MDAE	TEM
MDAE	D0TD

TE9LR:

LPHAN:

[TURN 0FF CHAR GEN
[TURN 0N THE VECTOR GEN

[TURN 0FF LP
[LOAD DXDY P0SIT BEING DRAWN INTO AR
[MASK ALL BUT LAST BIT-M/D BIT
[JUMP IF DRAW BIT SET

[LOAD CONTENTS 0F DXDY P0SIT INTO AR
[PUT P0SIT 0F SELECTED D0T IN DXDY

ARMED
MD10'9" L1

ZF:

ATBL:

ADD:

GR :

ROUTINES TO PICK
BITS (CONTD.)

```
COMBI:
SAVE
ERR
.
SAVE
XFLG
. +3
SAVE
INV
C6
BTS
COMBI:
SAVE
INV
.
SAVE
XFLG
. +3
SAVE
FFT
C6
C1
BTS
COMBI:
SAVE
FFT
.
SAVE
XFLG
. +3
SAVE
FFS
C6
C4
BTS
COMBI:
```

JPSR
MDAR
MDIR
JUMP
ARMD
MDAR
JPAN
MDAR
MDIR
MDAR
ARMD
JPSR
MDAR
MDIR
JUMP
ARMD
MDAR
JPAN
MDAR
MDIR
MDAR
MDAE
ARMD
JPSR
MDAR
MDIR
JUMP
ARMD
MDAR
JPAN
MDAR
MDIR
MDAR
MDAE
ARMD
JPSR

二五

二五五

五五

MDAR
MDIR
JUMP
ARMD
MDAR
JPAN
MDAR
MDIR
MDAR
MDAE
ARMD
JPSR
MDAR
MDIR
JUMP
MDIR
JUMP
ARMD
MDAR
JPAN
MDAR
MDIR
ARX0'F
ARMD
MDAR
MDAR'A
ARMD
MDAR'F
ARMD
MDAR'F
ARMD
MDAR'X
MDX0
JPLS
MDAR
MDIR

FFK:

SAV0
FFS
.
SAV0
XFLG
.+3
SAV0
FFK
C6
C6
BTS
C0YB1
SAV0
FFK
.
LNULL
.
SAV0
XFLG
.+3
SAV0
ERAS
.
ECNT
DXDY
C2
DXDYE
ETBL0-2
ZZE
TBL0
ZZZ
ECNT
C26
.+3
SAV0
ERAS

[ROUTINES TO PICK BITS
[CONTINUED

LNULL:
ERAS:

[DO=NOTHING ROUTINE
[FOR FUNC. SWITCHES
[ROUTINE TO ERASE A
[SELECTED ELEMENT

[MASK ID BITS
[L0AD ADDRESS OF LAST
[TBL0 LOCATION INTO ZZE
[L0AD STARTING ADDRESS OF
[TBL0 INTO ZZZ

MDAR'X	ZZZ	[INDEX ZZZ TO NEXT TBL0 ENTRY
MDAR'I	ZZZ	[LOAD CONTENTS OF ZZZ TO AR
MDAR'A	C2	[MASK ID BITS
MDX0	DXDYE	[AND COMPARE WITH DXDYE
JPLS	•-9•	
ARAR'F'H		
JPLS	•-11•	
MDAR	ZZE	[CHECK IF LAST ENTRY IN TBL0,
MDX0	ZZZ	[IF NOT JUMP AHEAD
JPLS	•+14•	[14 INST
MDAR	TBL0	[IF LAST ENTRY
MDAE'N	C1	[DECREMENT TBL0 PMINTER AND
ARMD	TBL0	[TCNT COUNTER
MDAR	TCNT	
MDAS'N	C1	
JPAN	•+2	
JUMP	•+2	
ARX0'F		
ARMD	TCNT	[RESTORE AR AND RETURN
MDAR	C1	
ARMD	XFLG	
MDAR	SAV0	
MDIR	ERAS	
MDAR	ZZZ	
MDAE	C1	
ARMD	T1	
MDAR'I	T1	
ARMD'I	ZZZ	
MDAR'X	ZZZ	
JUMP	•-22•	[ZZZ THEN STEP
JUMP	•	[THIS ROUTINE SETS THE
ARMD'0	CFLG	[EXIT FLAG
MDIR	CNCT	
JUMP	•	[ADDS TYPE OF GATE
ARMD	SAVX	[INT0 LOWER DIGITS OF
MDAR'X	TBL0	[TBL0 TABLE

CNCT:	
COMBI:	

MDAR	FLG1	TURN OFF IMAGE DONE FLAG
MDAE'N	C1	
JPAN	A1	JUMP TO DRAW ADDER
MDAE'N	C1	
JPAN	B1	JUMP TO DRAW ORRER
MDAE'N	C1	
JPAN	I1	JUMP TO DRAW INVERTER
MDAE'N	C1	
JPAN	FF1	JUMP TO DRAW TRIG FF
MDAE'N	C1	
JPAN	FF2	JUMP TO DRAW CS FF
MDAE'N	FF3	
JPAN	FF3	JUMP TO DRAW JK FF
MD07	DADD	
MDAR'L		
MD05	DADD	
ARM	77756	
MD05	DADD	STARTS DRAWING
JUMP	ENDRW	DRAW AND GATE
MD07	DORD	
MDAR'L		
MD05	DORD	
ARM	77756	
MD05	DORD	DRAW OR GATE
JUMP	ENDRW	
MD07	DINVD	
MDAR'L		
MD05	DINVD	
ARM	77756	
MD05	DINVD	DRAW INVERTER
JUMP	ENDRW	
MD07	FFTD	
MDAR'L		
MD05	FFTD	
ARM	77756	
MD05	FFTD	DRAW TOGGLE F-F

A1:

A1:

I1:

FF1:

FF2:	JUMP	ENDRW	[CONTINUE TO DRAW
	MD07	FFSD	
	MDAR'L		
	MD05	FFSD	
	ARM0	77756	
	MD05	FFSD	[DRAW SET-CLEAR F-F
	JUMP	ENDRW	
	MD07	FFKD	
	MDAR'L		
	MD05	FFKD	
	ARM0	77756	
	MD05	FFKD	[DRAW J-K F-F
	JUMP	ENDRW	
	MDAR	ASAV	
ENDRW:	JUMP'I	DRW	
	MDAR'F	TBL0	[THIS ROUTINE BUILDS THE
	MDAR'A	MASK1	[FLIP-FL0P DATA TABLE
	ARM0	TTEMP	
	MDAR'F	FFDT+1	[SET POINTERS TO TOP
	MDAR'A	MASK1	[0F TABLE
	ARM0	DTEMP	
	MDAR'X	TTEMP	[GET NEXT TBL9 ENTRY
	MDAR	TBL0	
	MDAE'L	1	
	MDX0	TTEMP	[JUMP OUT IF LAST
	JPLS	+2	[TBL0 ENTRY
	JUMP	FTE	
	MDAR'I	TTEMP	
	MDAR'A'L	7	
	MDAE'N'L	3	[SEE IF IT IS A F-F
	JPAN	FTRET	[GET NEXT IF NOT
	MDAR'I	TTEMP	
	MDAR'A'L	-100007	[CLEAR BITS 14,27,28,29
	ARM0'I	DTEMP	[STORE IN FFDT
	MDAR'X	DTEMP	[GET NEXT F-F ENTRY IN
	JUMP	FTRET	[TBL0


```

FTE:
MDAR
ARMD
JUMP'I
N99P
ARMD
MD06
ARX0'F
ARMD
MD11
MD07
MD10'A'L;
MDAR'L
MD05
ARMD
MD05
MDAR
JPLS
JUMP
MD10'A'L;
MDAR
JUMP'I
N99P
MDAR
ARMD
MD10'A'L;

DTRW:
DTEMP
FFDT
EXIT
ASAV
INT
LFLG
SCALD
D8TD
20VH
D8TD
77756
D8TD
LFLG
*+2
*-2
-20VH
ASAV
DTRW
C1
E8PFL
-1000VH

LPAN:
E8P:
[CONSTANTS AND DATA]
SAVAR:0
FSET:2
C1:1
FS:0
SAVG:61400VH
DXCY:7057600000
ASAV:0
LFLG:1
SAVDR:0
FCNT:0

[FFDT BUILT
[EXIT TO LIL
[Draws the GRID D8TS
[SET INTENSITY
[TURN OFF IMAGE DONE FLG
[SET SCALE
[TURN ON LP
[START THE DRAWING
[Draw the D8TS

[THIS PROGRAM EXIT IS NOT
[USED.

```


CONSTANTS AND VARIABLES, CONTD

ARSAV:0
 SCAL:07770JH
 INT:14000
 C2:7777677770
 TCNT:0
 GCNT:0
 FLG1:0
 T1:0
 C3:7
 C4:2
 EBPFL:0
 SAV9:0
 D1:6277661776
 D2:6677661776
 D3:7367661776
 C5:0
 XFLG:2
 SCLS:14000JH
 SAVTX:0
 D4:0400061776
 EXIT:\$DDEF
 CFLG:0
 DUN:0
 SCALD:34000JH
 AA1:77776JH
 A2:77776
 TEM:0
 D5:1010061776
 D6:1500061776
 C6:3
 BTS:0
 DXPYE:0
 ZZ:0
 ZZE:0
 SAVX:0
 ECNT:0

C26:32
 MASK1:77777
 TTEMP:0
 DTEMP:0
 FFDT:

[FLIP-FL0P DATA TABLE

0 REPEAT
 0 15.
 0 0

[GATE DISPLAY TABLE

0 REPEAT
 0 26.

[DATA T0 DRAW AND GATE

ETPL0:
 DADD:
 0000000000
 0000004000
 0400002001
 0400075777
 0000073777
 7377673777
 7377604001
 0000004001
 0400000000
 0600000001
 0000000000
 0000000001
 0000100000
 0000003400
 6137616400
 6137616401
 7057616400
 7057616401
 0000016400
 0000016401
 0720016400
 0720016401
 1640016400

[DATA T0 DRAW THE DOTS

D0TD:

CONTINUE DET DATA

1640016401
1640007200
1640007201
0720007200
0720007201
0000007200
0000007201
7057607200
7057607201
6137607200
6137607201
6137600000
6137600001
7057600000
7057600001
0000000000
0000000001
0720000000
0720000001
1640000000
1640000001
1640070576
1640070577
0720070576
0720070577
0000070576
0000070577
7057670576
7057670577
6137670576
6137670577
6137661376
6137661377
7057661376
7057661377
0000061376

0000061377	
0720061376	
0720061377	
1640061376	
1640061377	
1640055776	
6137655777	
0000100000	
DBRD: 0000000000	
0000004000	
0400002001	
0400075777	
0000073777	
7377673777	
7577675777	
7577602001	
7377604001	
0000004001	
0400000000	
0600000001	
0100001000	
0100076777	
0200000000	
0000000001	
0000100000	
DINV: 0000000000	
7377604000	
0400000001	
0600000001	
0400000000	
7377673777	
7377604001	
7377600000	
7177600001	
0000010000	
0000076777	

[DATA T8 DRAW SR GATE

[DATA T8 DRAW INVERTER

0050076777
7727676777
0050001000
7727601001
0000100000
0000000000
0100001000
0100076777
7677676777
7677601001
0100001001
0000100000
0000000000
0400004000
7377604001
7377673777
0400073777
0400004001
0400002300
0600002301
0400075476
0600075477
7377602300
7177602301
7377600000
7177600001
7377675476
7177675477
0000002000
0000075777
0200002000
7577602001
0000100000
0000000000
0400004000
7377604001

SQR:

FFTD:

FFSD:

[INVERTER DATA CONTD.]

[BASIC SQUARE DATA

[TOGGLE F-F DATA

[CLEAR-SET F-F DATA

7377673777
0400073777
0400004001
0400002300
0600002301
0400075476
0600075477
7377602300
7177602301
7377675476
7177675477
0200002000
7577602001
7577600001
0200000001
0200075777
7577675777
0000100000
0000000000
0400004000
7377604001
7377673777
0400073777
0400004001
0400002300
0600002301
0400075476
0600075477
7377602300
7177602301
7377675476
7177675477
7677602000
7677675777
0200002000
0000000001

FFKD:

ICLEAR-SET F-F DATA
ICONTINUED

IJK FLIP-FL0P DATA

[J-K F-F DATA CONTINUED
[TEXT DATA FOR THE MENU

0200075777
0000100000
P(22,11,29.,13)
P(102.,106,123,55)
P(62,11,39.,106)
P(123,55,63,11)
P(52.,106,123,55)
P(64,11,72.,106)
P(123,55,66,11)
P(82.,106,123,55)
P(67,11,94.,106)
P(123,55,70,0)
P(13,100.,11,29.)
P(101,116,104,11)
P(40.,117,122,11)
P(52.,11,116,126)
P(11,72.,124,106)
P(106,11,82.,103)
P(123,106,106,11)
P(94.,112,113,106)
P(106,11,62.,106)
P(123,55,65,13)
P(94.,11,58.,23)
P(105,122,101,123)
P(105,0,0,0,1)

[USER DEFINED COMMANDS

WD05=25000VH
WD06=26000VH
WD07=27000VH
WD10=30000VH
WD11=31000VH
TERMINATE

[END 9F LOGMM (DEFINE MODE).....

TXT:


```

EXPUNGE
TITLE C0NCT
ENTRY C0NCT,HDX,Y,HDX,Y1,DATA,DATA1,DATA2,C11,TEMP

C0NCT:
SKIP:
N00P
JUMP
JPSR
ARX0,F
ARMD
ARMD
JPSR
TABLE
MDAR
JPAN
MDAR,L,
MDAE,L
JUMP
ARIR
O REPEAT
O
ENDR
O REPEAT
O
ENDR
40000VH
40000VH
40000VH
O
40000VH
O
O
O
O

C11,TEMP
[THIS PROGRAM IMPLEMENTS CONNECT

*+1
CINIT
[URNS ON CURSOR
ECFLG
$ANFLG
$FIN
[00 CHECK FNS
[TIME TO EXIT
[YES, SO DO IT
[ELSE
[EXITS TO DDEF+2 TO DRAW
[FNS TABLE--MODE EXIT FNS-1

[ERASE A LINE

[FNS-9--CURSOR UP
[FNS-10--CURSOR LEFT
[FNS-11--CURSOR RIGHT
[FNS-12--SELECT VECTOR FWD
[FNS-13--CURSOR DOWN
[FNS-14--MOVE
[FNS-15--DRAW
CUP
CLEFT
CRITE
SVF
CDWN
MVEC
DVEC
CNULL
CNULL

```


[MOVE CURSOR LEFT

• HDXY
DDX
HDXY
CLEFT

JUMP
MDAR
MDAE'N
ARMD
MDIR

CLEFT:

[MOVE CURSOR RIGHT

• HDXY
DDX
HDXY
CRITE

JUMP
MDAR
MDAE
ARMD
MDIR

CRITE:

[MOVE CURSOR UP

• HDXY
DDY
HDXY
CUP

JUMP
MDAR
MDAS'N
ARMD
MDIR

CUP:

[MOVE CURSOR DOWN

• HDXY
DDY
HDXY
CDWN

JUMP
MDAR
MDAS'N
ARMD
MDIR

CDWN:

[SET MOVE VECTOR

• HDXY
MASK1
TEMP
COUNT
MVEC

JUMP
MDAR
MDAR'A
ARMD'X'I
MDAR'X
MDIR

MVEC:

[SET DRAW VECTOR

•

JUMP

OVEC:

MDAR	HDX	[ZER9S IN BITS 14 AND 30]
MDAR'A	MASK1	[ONE IN BIT 30]
MDAR'@	MASK2	
ARM'D'X'I	TEMP	
MDAR'X	COUNT	
MDIR	DVEC	
	0	[TURN CURSOR OFF]
JUMP	C1	
MDAR'X	S7	
MDIR		
	.	[TURN CURSOR ON
JUMP	C2	[ALSO SETS UP POINTERS IN
MDAR	C1	[DATA1 TO INITIALIZE
ARM'D	HDX	[CONNECT TABLE
ARM'D	DATA1-1	
MDAR'F	TEMP	
ARM'D	TEMP1	
ARM'D	TEMP2	
ARM'D	TEMP2	
MDAR'X'I	SKIP	
MDAR'X	CINIT	
MDIR		
	.	[SELECT VECTOR FORWARD
JUMP	TEMP1	
MDAR'X'I	MASK1	
MDAR'A	DATA2+1	
ARM'D	HDX	
ARM'D	TEMP2	
MDAR'X'I	DATA2+2	
ARM'D	C2	
MDAR	C11	
ARM'D	SVF	
MDIR		
	.	[ERASE VECTOR
JUMP		
ARX@'F		

ARM'D'I	TEMP2
MDAR'X	C11
MDIR	ERASV
SVB:	[SELECT VECTOR BACKWARD
JUMP	•
MDAR	TEMP1
MDAE	D1
ARM'D	TEMP1
MDAR'I	TEMP1
MDAR'A	MASK1
ARM'D	DATA2+1
ARM'D	HDX
MDAR	TEMP2
MDAE	D1
ARM'D	TEMP2
MDAR'I	TEMP2
ARM'D	DATA2+2
MDAR	C2
ARM'D	C11
MDIR	SVB
JUMP	•
ARM'D'0	ECFLG
MDIR	ENDCT
JUMP	•
MDIR	CNULL

[SET THE EXIT FLAG

[D9-N9THING ROUTINE

[CONSTANTS AND VARIABLES]

ECFLG:0

COUNT:1

HDX:0

HDX1:0

SAVEA:0

SCAL:37777JH

CONSTANTS AND VARIABLES
CONTINUED

INT:20000
DDX:50VH
DDY:50
MASK1:7777677776
MASK2:00000000001
C1:1
C2:0
THETA:0
ASAV:0
TEMP:0
TEMP1:0
TEMP2:0
SAVE:0
C11:1
D1:-1

DATA TO DRAW THE CURSOR

DATA: 00500000000
7727600001
0000000500
0000077277
0000000000
0000100000

TABLE OF CONNECTION LINES

DUMMY: 0000000000
DATA1: 0000000000
0000100000
0000100000
0000100000
0000100000
REPEAT 60.
0000100000
ENDR

TABLE TO REDRAW THE
SELECTED VECTOR

DATA2: 0000000000
0000000000
0000000000
0000100000

0000100000
0000100000

USER DEFINED INSTRUCTIONS

MD05=25000VH
MD10=30000VH
MD06=26000VH
MD07=27000VH
MD11=31000VH

TERMINATE

END OF CONCT (CONNECT MADE)

EXPUNGE

```
TITLE ANALR
ENTRY ANAL,RAWTP,RBP,RAWRT,RAWBK
ENTRY CANAL,RTCAL,LSRCH,TBSCH,CHKX,CHKY,BANAL,UNRES,D9AGN
ENTRY BAND,URD,LEVEL,STUFF,STUF,AB9V,BAND9
ENTRY TAEI,TAEIR,TAZE,BKSTF,E9LPD,ENDTA,FULD
ENTRY DT1P,NOT1E,BDT1,ENDCA,ANFLG,RIT
ENTRY RESP8,C0P8,NBSCH,NBSR1,NBSR2,C0MP,0UTP,0TSCB,SUNP
ENTRY 0PTR,NBAND,NBP,NBB8T,STACK,STKPT,STKBT,ID,9P,LNBP
ENTRY 9PNS,0TPTR,ANSW,BURD,CCNT,CHAR,CFLAG,URI,URIPT
ENTRY BTURI,N0UTS,ANFNS,ANXIT,ANRST,XITFG,RSTFG,ANCHK
ENTRY CMASK
```

```
ANAL:
N00P
MD10'A'UJ
ARX0'F
ARM0
ARM0
JPSR
MDAR
JPAR
MDAR
JPAR
N00P
MDAR
JPLS
JUMP
MDX0'F
JPLS
MDAR'X
MDAR'L
JUMP
ARM0
MDAR'F

-1000VH
XITFG
RSTFG
$FIN
ANFNS
XITFG
$DDEF
RSTFG
$DR3
ANFLG
AN1
RIT
1
AN2
ANFLG
$DDEF2
$WT1
$CTAB

[SHUT OFF THE F/C
[CLEAR EXIT AND RESET
[FLAGS
[GO CHECK ANALR FNS
[SEE IF EXIT 9R RESET
[GO T9 MAIN DISPLAY MODE
[GO T9 SET UP FOR NUMBS AGAIN
[CHECK-SEE IF ANALYSIS NEEDS
[T9 BE DONE FOR FIRST TIME
[IF S9 RE-INITIALIZE TABLES
[IF ANFLG=1 SET UP FOR
[ACCEPTING CHARACTERS AND
[INCREMENT ANFLG
[SET UP PIVOTS IN AMRMX FOR
[TELETYPE INTERFACE
```

ANCHK:

AN1:

1A1:	MDAE	C1	[SET POINTER TO NEXT
	MDAR'A	MASK1	[CTAB ENTRY TO BE
	ARM	\$CTAB	[FILLED
	ARX0'F		
	ARM	CCNT	[CLEAR SOME FLAGS
	ARM	CFLAG	
2A1:	JPSR	\$ICHTY	[GO GET A CHAR FROM THE TTY
		2	
	MDBR	\$TTYC	
	MDER'A	CMASK	[BRING IT INTO BR, MASK AND
	BRMD	CHAR	[STORE IN CHAR
	ARM0'0	CFLAG	
	JUMP	2A1	
AN2:	MDX0'F	1'2	[DO NOT DO ANYTHING UNTIL
	JPLS	AN3	[LIL INCREMENTS ANFLG AFTER
	JUMP	\$DDEF2	[ALL LABELS ARE IN
AN3:	MDX0'F	2'3	
	JPLS	AN4	[SET UP TO ACCEPT NUMBERS
	MDAR'X	ANFLG	
	MDAR'F	\$NTAB	
	MDAE	C1	
	MDAR'A	MASK1	
	ARM	\$NTAB	
1A3:	ARX0'F		
	ARM	CCNT	
	ARM	CFLAG	
	JUMP	\$DDEF2	[JUST GO DRAW UNTIL ALL
AN4:	MDX0'F	3'4	[NUMBERS ARE IN
	JPLS	AN5	
	JUMP	\$DDEF2	
AN5:	MDX0'F	4'5	
	JPLS	AN6	[ALL NBS. IN...
	JPSR	RESP0	[GO CALCULATE RESPONSE
	JUMP	\$DDEF2	
AN6:	MDX0'F	5'6	
	JPLS	.	[SHOULD NOT GET HERE

RIT:	JUMP	\$DDEF2	[THIS ROUTINE RE-INITIALIZES
	MDAR'F	RAWBK	[THE BAND,BANDS,AND URD TABLES
	MDAE'N	C1	[TO ZERO
	MDAR'A	MASK1	
	ARM	CLRFG	[GET NEXT DATA ENTRY
CLRR:	MDAR'X	CLRFG	[CHK AGAINST BOTTOM OF DATA
	MDX0'F	BTAB	
	JPLS	•+2	
	JUMP	TAEI	[JUMP OUT IF RIT COMPLETE
	ARX0'F		
	ARM'D'I	CLRFG	[CLEAR OUT DATA ENTRY
	ARM	NOUTS	
	JUMP	CLRR	[RETURN TO CLEAR NEXT ENTRY
	MDAR'F	\$DATA1	
TAFL:	MDAE'N	C1	[THROW AWAY ERASED LINES
	MDAR'A	MASK1	
	ARM	DT1P	[PTR TO LAST NON-ZERO DATA1 ENTRY
	ARM	NDT1E	[PTR TO NEXT ENTRY TO BE CHKED
	ARX0'F		
	ARM	ZFLAG	[NO ZEROS FOUND FLAG
	MDAR	\$TEMP	[GET BOTTOM OF DATA1 SET FM CONCT
	ARM	BDT1	[BOTTOM POINTER
	MDAR'X	BDT1	[SET TO ENTRY BELOW DATA1
	MDAR'X	DT1P	
	MDX0	BDT1	[LOOK AT NEXT DATA1 ENTRY AND
	JPLS	•+2	[CHECK AGAINST BOTTOM PTR
	JUMP	EOLPD	[BRANCH OUT TO FIXUP UP END OF TABLE
	MDAR'I	DT1P	
	JPLS	TAEI	
	ARAR'H		
	JPLS	TAEI	[NON-ZERO ENTRY FOUND--RETURN
	MDAR	DT1P	
	ARM	NDT1E	
	ARM'D'0	ZFLAG	[ZERO ENTRY FOUND--SET PTRS AND FLAG
	MDAR'X	NDT1E	[THROW AWAY ZERO ENTRY
TAZE:	MDX0	BDT1	[GET NEXT NONZERO ENTRY WITH NDT1E

JPLS	•+2	[BRANCH OUT IF AT BOTTOM OF TABLE
JUMP	E0LPD	
MDAR'I	NDT1E	[JUMP TO SWAP ENTRIES
JPLS	BKSTF	
ARAR'H		
JPLS	BKSTF	
JUMP	TAZE	[MUST BE ZERO ENTRY--GET NEXT ONE
MDAR'I	NDT1E	[ROUTINE TO STUFF A NONZERO ENTRY
ARMD'I	DT1P	[INTO A PREVIOUS ZERO ENTRY
ARXG'F		
ARMD'I	NDT1E	[CLEARS OUT FOUND NONZERO ENTRY AND
JUMP	TAELR	[RETURNS
MDAR'N	ZFLAG	[ROUTINE TO PAD OUT TABLE WITH E0L'S
UPAN	ENDTA	[DO NOT DO IF NO ZERO ENTRY FOUND ABOVE
MDAR	NDT1E	
MDX0	DT1P	
JPLS	•+2	[CHECK AGAINST LAST TABLE ENTRY
JUMP	ENDTA	
MDAR'H	C1	
ARMD'I	DT1P	[STORE AN E0L INTO THIS ZEROED ENTRY
MDAR'X	DT1P	
JUMP	ELR	[RETURN TO E0L PAD NEXT ENTRY
MDAR'F	\$DATA1	
MDAE'N	C1	
MDAR'A	MASK1	
ARMD	RAWTP	[SET UP PTR IN DATA1 SET
MDAE'N	C1	
ARMD	LRWTP	
MDAR'F	RAWBK	
ARMD	RBP	[SET UP PTR IN NEW 2-CONNECT SET
MDAR	\$DATA1	
MDAR'A	C1	
JPLS	•	[A TRAP IF 1ST DATA1 ENTRY IS DRAW
MDAR'X	LRWTP	
MDAR'X'I	RAWTP	
MDAR'A	C1	

JPLS MDAR'I'H
 MDAR'A
 JPLS MDAR'I
 MDAR'A
 JPLS MDAR'I
 ARMD'I
 JUMP MDAR'X
 MDAR'I
 ARMD'I
 MDAR'X
 MDAR'I
 ARMD'I
 JUMP MDAR'I
 MDAR'A
 JPLS
 JUMP MDAR'X
 MDAR'I
 ARMD'I
 MDAR'X
 MDAR'I
 ARMD'I
 MDAR'X
 MDAR'A
 ARMD
 MDAR'F
 MDAR'N
 MDAR'A
 ARMD
 MDAR'F
 MDAR'A
 ARMD
 MDAR'F

RAWRT
 RAWTP
 C1
 FULD
 LRWTP
 C1
 DRFD
 RAWTP
 RBP
 RAWRT
 RBP
 LRWTP
 RBP
 RBP
 RAWTP
 RBP
 RAWRT
 LRWTP
 C1
 •+2
 CANAL
 RBP
 LRWTP
 RBP
 RBP
 MASK1
 BTRBK
 RAWBK
 C1
 MASK1
 TPRBK
 BAND
 MASK1
 BPB
 BAND0

DRFD:

FULD:

CANAL:

[CHK IT- JUMP IF A DRAW
 [A MOVE--CHK E0L
 [2-C COMPLETE G0T9 CHK CONNECT

[STORE LAST DRAW IN 2-C SET

[STORE THE MOVE IN 2-C SET
 [GET NEXT ENTRY
 [FIX UP LAST DRAW
 [IF LAST DATA1 ENTRY IS DRAW
 [STORE IT AS END 9F CONNECT
 [FROM THE PREVIOUS MOVE
 [JUMP CANAL EXECUTED IF LAST
 [ENTRY IS A MOVE, OTHERWISE
 [FALL THROUGH...

[SET PTR TO ONE BELOW RAWBK
 [I. E. THE BOTTOM

[SET PTR TO T9P 0F RAWBK

[PTR TO CURRENT ENTRY IN BANAL TABLE

MDAR'A
ARMD
MDAR'F
MDAR'A
ARMD
MDAR'F
MDAR'A
ARMD
MDAR
ARMD
MDAR'X
MDXG
JPLS
MDAR
MDAE
ARMD
MDAE'N
ARAR'H
JPAR
JUMP
MDAR'I
ARAR'H
MDAR'A
JPLS
MDAR'I
ARMD
MDAE'N'H
MDAE'N
JPAR
JUMP
MDAR'F
MDAR'A
ARMD
MDAR

RTCAL:

LSRCH:

TBSCH:

MASK1
BPB0
URD
MASK1
BURD
URI
MASK1
URIP
LEVLL
LEVEL
TPRBK
RBP
RBP
BTRSK
LSRCH
LEVEL
LEVIN
LEVEL
LEVEL
RTCAL-2
ENDCA
RBP
C1
RTCAL
RBP
FDRBK
LEVEL
LEVEL
+2
RTCAL
\$TBL0
MASK1
TBPT
\$TBL0

[PTR TO CURRENT ENTRY IN BANDS TABLE

[INIT. BOT 0F URD TO TOP 0F URD

[INIT. LEVEL TO LEFT BETWEEN C0LS 1 2

[(RE)INIT. PTR TO CURR. RAWBK ENTRY

[INCR. AND CHK IF LAST RAWBK ENTRY
[JUMP IF NOT

[INCR LEVEL
[CHK IF LEVEL HAS BEEN SEARCHED

[RE-INITIAL RBP PTR AND D0 NXT LEVEL
[ALL LEVELS CHKD, EXIT CANAL

[CHK E0L SEE IF ENTRY ALRDY MATCHED

[CHK IF RAWBK ENTRY IS AT THIS LEVEL

[LOAD CURR TBL0 PTR

MDAE
MDAR'A
ARMD
ARX0'F
ARMD
ARMD
MDAR'X
MDX0
JPLS
JUMP
MDAR'I
MDAE'N
ARMD
JUMP
ARAR'N
ARMD'0
JPAN
ARAR'H
MDAR'A
MDAE'N
JPAN
JUMP
MDAR'H
JUMP
ARAR'N
ARMD'0
JPAN
ARAR'H
MDAR'A
ARMD
MDAE'N
JPAN
JUMP
MDAR'I
MDAR'A
ARMD'I

RTBS:

CHKX:

CHKY:

C1
MASK1
TBB0T

UPFLG
0TFLG
TBPTR
TBB0T
•+2
UNRES
TBPTR
FDR0K
COMDIF
•+3

0TFLG
•-2

MASK1
ADX2
•+2
RTBS
COMDIF
•+3

UPFLG
•-2

MASK1
YDIFF
ADY2
•+2
RTBS
RBP
RXYM
RBP

[LOAD BOTTOM 0F TBL0

[CLEAR SOME FLAGS

[CHKD WHOLE TABLE IF S0 UNRESOLVED

[A WORD CONTAINING X Y DIFF FM GATE

[YIELDS ABS VAL 0F X AND SETS 0TFLG
[0TFLG IS MINUS ZERO IF CONNECT POINT
[IS T9 THE LEFT 0F GATE

[LEAVE IF CONNECT PT T00 FAR FM GATE

[SIMILAR T9 ABOVE FOR Y DIFF
[UPFLG IS MINUS ZERO IF CONNECT
[POINT IS ABOVE CENTER 0F GATE

[CLEARS 0UT X,Y FROM RAWBK
[WORD FOR FIXUP

BANAL:

MDAR'I
MDAR'A'L;
MDAR'0'I
ARM0'I
MDAR'A'L;
ARAR'H
ARM0'I
MDAR'I
MDAR'A'L;
MDAR'0'I
ARM0'I
MDAR'A'L;
AR0S
MDAR'0'I
ARM0'I
MDAR'I
MDAR'A
ARM0
MDAR'0'I
ARM0'I
MDAR
MDX0'F
JPLS
MDAR
MDAE'N'L;
JPAN
JUMP
MDAR'I
MDAR'0'L;
ARM0'I
JUMP
MDAR'N
JPAN
MDAR'I
MDAR'0'L;
ARM0'I

[BUILDS ANALYSIS DATA BLOCKS
[FIXUP RAWBK X
[STUFF BAND X
[FIXUP RAWBK Y
[THIS SECTION EXTRACTS 6 BIT REPN
[9F X VALUE 0F GATE,Y VALUE 0F GATE
[AND TYPE 0F GATE AND STUFFS IT INTO
[BANAL TABLE

[CHECK SEE IF THIS IS A F-F
[IF S9 FIND THE MIDDLE INPUT

[AND LABEL IT WITH A 2 IN PTR

[CHK JPFLAG. SET BIT TWO IF =0
[B9TT9M 0F GATE

UCHK:

STUFF:	MDAR	RBP	[STUFF A POINTER TO THE CONNECTED
	ARM	CPT	ENTRY IN RAWBK INTO THE UPPER
	MDAR'I	RBP	[HALF WORD
	MDAR'A	C1	
	JPLS	AB9V	
	MDAR'X	CPT	[CONNECTED WORD BELOW CURRENT WORD
	JUMP	STUF	
	MDAR	CPT	[CONNECTED WORD ABOVE
AB9V:	MDAE'N	C1	
	ARM	CPT	
	MDAR'H	CPT	
STUF:	MDAR'0'I	BPB	
	ARM'I	BPB	
	MDAR'N	0TFLG	
	JPN	N9R9T	
	MDAR'H'I	BPB	[BANAL WORD IS BUILT IN LOWER HALF
	ARM'I	BPB0	[WORD. IF AN OUTPUT, HALF-WORD
	AR0'F	BPB	
	AR'D'I	BPB0	
	MDAR'X	+2	[ROTATE AND STORE IN BAND0
	JUMP	BPB	[ROTATION NOT REQUIRED (INPUT)
N9R9T:	MDAR'X	00AGN	
	JUMP	URD	
UNRES:	MDAR'F	C1	[UNRESOLVED ENTRY, UNABLE
	MDAE'N	MASK1	[TO MAP TO A GATE
	MDAR'A	UNRPT	
	ARM	UNRPT	[CENTER IN URD TABLE
1UNR:	MDAR'X	BURD	
	MDX0	+2	
	JPLS	2UNR	
	JUMP	UNRPT	[CHECK IF ANOTHER URD ENTRY
	MDAR'I	FDRBK	[HAS SAME COORDINATES
	MDX0	1UNR	[IF S9 BUILD SIMILAR URI ENTRY
	JPLS	BUR	[RETURN TO LOOK AT NEXT RAWBK
	JPSR	00AGN	
	JUMP		

2UNR:	MDAR	FDRBK	[IF THIS URD IS NOT FOUND THEN BUILD
	ARMD'I	UNRPT	[A NEW URI ENTRY
	JPSR	BURI	
	MDAR'X	UNRPT	
	ARMD	BURD	
	MDAR'H	C1	[SET E9L FOR THIS CONNECT WORD (IT WAS
DBAGN:	MDAR'0'I	RBP	[CHECKED) RETURN TO CHK NXT CNT WORD
	ARMD'I	RBP	
	JUMP	RTCAL	
	MDAR'X	ANFLG	[CALL ANALYSIS TABLES BUILT INCREMENT
ENDCA:	MDAR	URIPT	[ANFLG AND GO GET LABELS
	ARMD	BTURI	
	JUMP	\$DDEF2	
		.	
BURI:	JUMP	UNRPT	[THIS SEGMENT BUILDS A URI ENTRY
	MDAR		
	ARAR'H	URIPT	[STORE URD ADDRESS IN UPPER URI
	ARMD'I	UNRPT	
	MDAR'I	C1	[TEST IF THIS URD IS AN OUTPUT
	MDAR'A	*+2	
	JPLS	1BUR	[IF S9 SET BIT 0 IN URI TABLE
	JUMP	NBUTS	[TO S9 INDICATE
	MDAR'X	40000VH	
	MDAR'L;	URIPT	[GET LAST TWO DIGITS OF RAWBK ADDR
	MDAR'0'I	URIPT	
	ARMD'I	RBP	
	MDAR	C77	[INDEX INTO ANSW
1BUR:	MDAR'A	ANSW	
	MDAS'F	MASK1	
	MDAR'A	URIPT	[STORE ANSW PINTER IN URI
	MDAR'0'I	URIPT	
	ARMD'I	URIPT	
	MDAR'X	BURI	[RETURN TO CALL
	MDIR		

ROUTINE TO SIMULATE CIRCUIT

RESP0:	JUMP	0
	MDAR'F	BAND
	MDAE'N	C1
	MDAR'A	MASK1
	ARMD	OPTR
	MDAR'F	NBAND
	MDAR'A	MASK1
	ARMD	NBP
	MDAR'X	OPTR
	MDX0	BPB
	JPLS	+2
	JUMP	NBSCH
	MDAR'I	OPTR
	ARMD'I	NBP
	MDAR'X	NBP
	JUMP	CEPB
	MDAR	NBP
	ARMD	NBB0T
	MDAR'F	NBAND
	MDAR'A	MASK1
	ARMD	NBP
	MDAR'F	STACK
	MDAE'N	C1
	MDAR'A	MASK1
	ARMD	STKPT
	MDAR'X	STKPT
	MDX0'F	STK3T
	JPLS	+2
	JUMP	NBGO-3
	ARX0'F	
	ARMD'I	STKPT
	JUMP	ZSTK
	MDAR'F	STACK
	MDAR'A	MASK1
	ARMD	STKPT
	MDAR'I	NBP
NBGO:		

FIRST MAKE A COPY OF BAND
FOR THIS ANALYSIS

SET UP STACK POINTER

NOW CLEAR OUT THE STACK


```

MDAR'A
ARMD
MDAR'A
ARMD
MDAR'I'H
MDAR'A
MDAE'L
MDAR
ARIR
ARMD'I
MDAR'I
MDAR'A'L'
MDAR'0'I
ARMD'I
MDAR'X
MDAR
ARMD
MDAR'X
MDX0
JPLS
JUMP
MDAR'I
JPLS
JUMP
MDAR'A
MDX0
JPLS
MDAR'I'H
MDAR'A
MDAE'L
MDAR
ARIR
ARMD'I
MDAR'I
MDAR'A'L'
MDAR'0'I

```

```

MASK1
ID
C7
0P
NBP
C77
ANSW
STKPT
NBP
30000H
STKPT
STKPT
STKPT
NBP
LNBP
NBP
NBS0T
*+2
GTLFF
NBP
*+2
NBSR2
MASK1
ID
NBSR2
NBP
C77
ANSW
STKPT
NBP
30000H
STKPT

```

NBSR2:

```

[SAVE GATE IDENTIFIER

[SAVE GATE TYPE

[LOAD VALUE 0F PR0PER ANSW L0C.
[STORE IN STACK
[EXTRACT HEIGHT 0F THE INPUT
[STORE THIS IN THE STACK F0R
[FLIPFLOP CALCULATIONS

[SEE IF WE VE LOOKED AT ALL ENTRIES
[IF S9 G0 GET LAST F-F VALUES
[IF N0T CONTINUE NBAND SEARCH

[GET NEXT ANSW VALUE F0R
[SAME ID ENTRY

```


MDAE'N	C1	[FIND ALL OUTPUTS OF THIS
MDAR'A	MASK1	[GATE AND STUFF THE RESULT
ARMD	0TPTR	[INT0 THE PROPER ANSW L9C.
MDAR'X	0TPTR	
MDX0	BPB0	
JPLS	•+2	
JUMP	X0JT	
MDAR'I'H	0TPTR	
MDAR'A	MASK1	
MDX0	ID	
JPLS	0TSCH	
MDAR'I	0TPTR	
MDAR'A	C77	
MDAS'F	ANSW	
MDAE'L	0	[SAME ID FOUND
ARMD	•+2	
ARMD	STACK	
MDAR	0	[ANSWER IS STUFFED INTO TW0
0	•-1	
MDAR	C1	[ANSW LOCATIONS
MDAE'N	•+2	
ARMD	STACK	
MDAR	0	
0	0P	
MDAR	3	[IF THIS SIMULATED A F-F
MDAE'N'L	0TSCH	
JPAN	0TPTR	
MDAR'I	10000	
MDAR'A'L	0TSCH	[DETERMINE LEVEL OF OUTPUT
JPLS	STACK	
MDAR'N	C1	
MDAR'A	A1	
MDIR	A2	[AND SET 0R RESET APPROPRIATELY
MDIR	0TSCH	
JUMP	0P	
MDAR		

0TSCH:

A1:

A2:
LFFCP:

XOUT:

MDAE'N'L'	3	SUNP	
JPAN		XTP	
MDAR'I		-100001	
MDAR'A'L'		XTP	[SAVE F-F DISPLAY COORDS
ARMD'I		STACK	
MDAR		100001	
MDAR'A'L'		XTP	[R IN NEW F-F VALUE
MDAR'0'I		XTP	
ARMD'I		LNBP	[IF ALL NBAND ENTRIES
MDAR'X		NBBET	[SIMULATED
MDX0		•+2	
JPLS		RESP0	[RETURN T0 RESP0 CALL
JUMP'I		LNBP	[ELSE
MDAR'I		•+2	
JPLS		SUNP	
JUMP		LNBP	
MDAR		NBP	
ARMD		NBSR1	
JUMP			[G0 GET NEXT TYPE 0F GATE
N00P			[SIMULATES AN AND GATE
MDAR'X		SP1	
MDX0		STKPT	
JPLS		•+2	
JUMP'I		SAND	
MDAR'I		SP1	[RETURN T0 CALL
MDAR'A		STACK	
ARMD		STACK	
JUMP		SAND+1	
N00P			[SIMULATES AN 0R GATE
MDAR'X		SP1	
MDX0		STKPT	
JPLS		•+2	
JUMP'I		S0R	[RETURN T0 CALL
MDAR'I		SP1	
MDAR'0		STACK	

SINV:	ARMD JUMP	STACK S0R+1	[SIMULATES INVERTER
	N00P		
	MDAR'N	STACK	
	ARMD	STACK	
	JUMP'I	SINV	[RETURN TO CALL
SCSFF:	JUMP	•	[SIMULATES SET-CLEAR F-F
	MDAR	STACK	
	MDAR'A	STACK+1	
	MDAR'A	C1	
	JPLS	SCSUD	[IF B9TH INPUTS SET••UNDEF
	MDAR	STACK	
	MDAR'0	STACK+1	
	MDAR'A	C1	[IF NEITHER INPUT SET
	JPLS	SC9N	
	MDAR	STACK-1	[LOAD LAST F-F VALUE
	ARMD	STACK	
	MDIR	SCSFF	[RETURN TO CALL
	MDAR'H	STACK+1	[ONE 9F THE INPUTS SET
SCN:	MDAR'A'L'	10000	
	JPLS	•+2	
	JUMP	SCSXT	[IF S +1 IS RESET TERMINAL
	MDAR	STACK+1	[THEN STACK ALREADY 0• K•
	ARMD	STACK	[ELSE STORE SET INPUT INTO
	JUMP	SCSXT	[TOP 9F STACK
	MDAR'H	C1	
SCSUD:	ARMD	STACK	[SET UNDEF• FLAG
	MDIR	SCSFF	[RETURN•
	MDAR	STACK	[WE SET SOMETHING
SCSXT:	MDAR'A'L'	-100000	[S0 CLEAR UNDEF FLAG
	ARMD	STACK	
	MDIR	SCSFF	[RETURN•••
	JUMP	•	[SIMULATE TOGGLE F-F
STFF:	MDAR	STACK	

MDAR'A	SMASK	
ARMD	STACK	
MDAR	STACK+1	
MDAR'A	SMASK	[MASK OUT ALL OF LOWER PART
ARMD	STACK+1	[OF WORD EXCEPT F-F VALUE
MDAR	STACK+2	
MDAR'A	SMASK	
ARMD	STACK+2	
MDAR	STACK	
MDAE	STACK+1	
MDAE	STACK+2	
ARMD	TSUM	
JPLS	1STFF	[NON ZERO SUM JUMP
MDAR	STACK-1	[NO INPUT FOUND, SO LOAD
ARMD	STACK	[LAST F-F VALUE AND RETURN
MDIR	STFF	
MDX0'F	1	[IS IT A SINGLE INPUT
JPLS	STFUD	[IF NOT THEN UNDEFINED
MDAR'F	STACK-1	
ARMD	SP1	
MDAR'I'X	SP1	[FIND OUT WHICH INPUT HAS
JPLS	•+2	[THE ONE ENTRY
JUMP	•-2	
MDAR'I'H	SP1	
MDAR'A'L'	30000	
JPLS	1FND	
ARX0'F	STACK	[ITS THE RESET LINE S0
ARMD	STFF	[RESET AND
MDIR	SP1	[RETURN TO CALL
MDAR'I'H	20000	
MDAR'A'L'	2FND	
JPLS	C1	[ITS A SET INPUT S0
MDAR	STACK	[SET TOP OF STACK AND
ARMD	STFF	[RETURN
MDIR	STACK	[ITS A TOGGLE INPUT
MDAR		

1STFF:

1FND:

2FND:

MDAR'A'L;	-1	[GET LAST F-F VALUE
ARMD	STACK	[COMPLEMENTED
MDAR'N	STACK-1	[9R STACK INT9 THIS
MDAR'A	C1	[SINCE MAY BE UNDEF
MDAR'0	STACK	
ARMD	STACK	
MDAR'H	STACK-1	
MDAR'A	C1	
ARAR'H		
MDAR'0	STACK	[SET UNDEF FLAG IF
ARMD	STACK	[ITS SET LAST TIME
MDIR	STFF	[RETURN TO CALL
MDAR'H	C1	[SET UNDEF FLAG FOR
MDAR'0	STACK	[ILLEGAL COMB9
ARMD	STACK	
MDIR	STFF	[RETURN TO CALL
JUMP	.	[SIMULATE J-K F-F
MDAR	STACK	[SIMILAR TO CLEAR-SET
MDAR'A	STACK+1	
MDAR'A	C1	
JPLS	JT9G	
MDAR	STACK	[EXCEPT TWO INPUTS ARE
MDAR'0	STACK+1	[LEGAL
MDAR'A	C1	
JPLS	JK	
MDAR	STACK-1	
ARMD	STACK	
MDIR	SJFFF	
MDAR'H	STACK+1	[SETS 0R RESETS THE F-F
MDAR'A'L;	10000	
JPLS	•+2	
JUMP	JKXIT	
MDAR	STACK+1	
ARMD	STACK	
JUMP	JKXIT	
MDAR'N	STACK-1	[BOTH INPUTS PRESENT

STFUD:

SJFFF:

JK:

JTGG:


```

[00 SET IN COMPLEMENT
[00 LAST F-F VALUE

[FNS TABLE--EXIT-FNS1

[RESET--FNS-4

[00-NOTHING ROUTINE

[RESET FNS PRESSED
[SET ANFLG TO 2
[AND SET RESET FLAG

[MODE EXIT PRESSED
[SET EXIT FLAG

[00 CLEAR ANFLG TO 00
[TO MAIN DISPLAY MODE

```

```

C1
STACK
SJKFF
STACK
-100000
SJKFF
ANXIT
ANULL
ANULL
ANULL
ANRST
13.
ANULL
.
ANULL
.
2
ANFLG
RSTFG
ANRST
.
XITFG
ANFLG
ANXIT

```

```

MDAR'A
ARMD
MDIR
MDAR
MDAR'A'L;
MDIR
0
0
0
0
0
REPEAT
0
ENDR
JUMP
MDIR
JUMP
MDAR'L;
ARMD
ARMD'0
MDIR
JUMP
ARMD'0
ARXB'F
ARMD
MDIR

```

```

JKXIT:
ANFNS:
ANULL:
ANRST:
ANXIT:

```

[CONSTANTS AND VARIABLES]

```

ANFLG:0
ZFLAG:0
CLRFG:0
C1:1
C7:7
C77:77
MASK1:77777
RXYM:0077700777
CPTR:0

```



```

RAWTP:0
RBP:0
BTRBK:0
TPRBK:0
BPS:0
RPSB:0
TBPTIR:0
TBRST:0
XITFG:0
RSTFG:0
RAWBK:

```

[MORE CONSTANTS AND VARIABLES

[TABLE OF CONNECTED PAIRS
[FROM DATA1

```

0
0
0
REPEAT 60.
0
ENR
0
0
REPEAT 40.
0
ENR
0
0
REPEAT 20.
0
ENR
0
0
REPEAT 20.
0
ENR
0
0
REPEAT 20.
0
ENR
0
0
REPEAT 20.
0
ENR
0
0
REPEAT 20.
0
ENR

```

[TABLE OF INPUT CONNECTS

```

0
0
0
REPEAT 40.
0
ENR
0
0
REPEAT 20.
0
ENR
0
0
REPEAT 20.
0
ENR
0
0
REPEAT 20.
0
ENR
0
0
REPEAT 20.
0
ENR

```

[TABLE OF OUTPUT CONNECTS

```

0
0
0
REPEAT 20.
0
ENR
0
0
REPEAT 20.
0
ENR
0
0
REPEAT 20.
0
ENR
0
0
REPEAT 20.
0
ENR

```

[TABLE OF UNRESOLVED DATA

```

0
0
0
REPEAT 20.
0
ENR
0
0
REPEAT 20.
0
ENR
0
0
REPEAT 20.
0
ENR
0
0
REPEAT 20.
0
ENR

```

[TABLE OF UNRESOLVED INPUTS
[AND OUTPUTS

```

0
0
0
REPEAT 20.
0
ENR
0
0
REPEAT 20.
0
ENR
0
0
REPEAT 20.
0
ENR
0
0
REPEAT 20.
0
ENR

```


TABLE OF ANSWER VALUES

ANSW: 0 0 REPEAT 63.

```

BTAB:      0      [BOTTOM OF DATA TABLES]
          0      [COPY OF BAND TABLE FOR]
          0      [ONE TIME ANALYSIS]

```

```

LASTK:
STACK:
O
O
O
REPEAT 10.
TABLE OF VALUES FOR
SIMULATION ROUTINES TO
OPERATE ON

```

STKBT:	0	
OPNS:	SAND	TABLE OF SIMULATION ROUTINES

INCRE CONSTANTS AND VARIABLES

```
UNRPT:0
LEVEL:0
LEVLL:65076
LEVLR:22200
LEVIN:7200
FERBK:0
```


[MORE CONSTANTS AND VARIABLES

```

CENDIF:0
RTFLG:0
UPFLG:0
ADX2:1200
ADY2:1100
DT1P:0
NDT1E:0
BDT1:0
LAST:0
LRWTP:0
CPTR:0
NBP:0
NRGT:0
STKPT:0
ID:0
AP:0
LNBP:0
STPTR:0
SP1:0
BURD:0
CCNT:0
CFLAG:0
CMASK:177
CHAR:0
UKIPT:0
RTURI:0
NGUTS:0
YDIFF:0
GTVE:0
XTP:0
CM77:77000
SMASK:777700001
TSUM:0

```

MD05=25000JH [USER DEFINED INSTRUCTIONS

MD06=26000JH
MD07=27000JH
MD10=30000JH
MD11=31000JH
ARC7=27100JH

TERMINATE

[END OF ANALR (ANALYSIS MBDE).....


```

EXPUNGE
TITLE INTR0
ENTRY INTR0,IFCLR,IE0LR,INFNS,IEEXIT,NPAGE,IRITE
ENTRY PGTAB
MACR01
P(A1,A2,A3,A4,A5)
A1VBVK + A2VB + A5VBVK + A3VBVK + A4VB + 0
ENDM
ZZZ=101
[ASCII CODE DEFINITIONS
(A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z)
IREPEAT ZZ, (A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z)
ZZ\.=ZZZ
ZZZ=ZZZ+1
ENDI
P.=120

INTR0:
ARX0'F
ARMD
ARMD
MDAR'F
ARMD
MD10'0'L,
JPSR

MDAR
MDAS'F
MDAR'A
MDAE'L
MDAR
ARIR'F
ARMD
JPSR

MDAR'N
JPAN
MDAR
JPAN
JUMP

IARG:
IARG
IRITE
0
DUNF
*-1
IXFLG
$DDEF
INTR0

IXFLG
DUNF
IFCLR
77755
61400JH
$FIN
INFNS
PAGE
PGTAB
MASK1
0
IARG
IRITE
0
DUNF
*-1
IXFLG
$DDEF
INTR0

[STORE PAGE ADDR IN IARG
[G0 WRITE THIS PAGE
[WAIT TO FINISH
[CHECK EXIT
[IF S9 G0 TO MAIN DISP MODE
[ELSE G0 WRITE PAGE AGAIN

```


IEGLR:	N9GP	•	ISAV	END 9F LIST HANDLER
	ARMD		ITFLG	
	ARMD'0		ISAV	
	MDAR		IEGLR	
	JUMP'I	•		FRAME CLOCK HANDLER
	N9GP			
	FPRI			
	ARMD		IFSAV	
	MDAR'X		IFCNT	DONE WITH TEXT WRITE
	MDAR'N		ITFLG	N0, RETURN
	JPAN		IFCND	
	MDAR		IFSET	RIGHT N0, 0F F/C TICKS
	MDX0		IFCNT	N0, RETURN
	JPLS		IFCND	
	ARMD'0		DUNF	YES, RESET F/C COUNT
	ARX0'F			AND REFRESH
	ARMD		IFCNT	
	MDAR		IFSAV	UNFREEZE INTERRUPTS AND
	UPRI			CLEAR INTERRUPT AND RETURN
	JUMP'I		IFCLR	FNS TABLE--MODE EXIT FNS-1
	0		IEKIT	
	REPEAT	14.		
	0		NPAGE	ANY OTHER FNS, THEN
	ENDR			
	0		INULL	
	0		INULL	
	0		NPAGE	INCREMENT PAGE COUNT
	0			SET EXIT FLAG
IEKIT:	JUMP	•	IXFLG	
	ARMD'0		IEKIT	
	MDIR	•		INCREMENT PAGE COUNT
	JUMP			
	MDAR'X		PAGE	CHECK AGAINST MAX PAGE N0.
	MDX0		PLIM	
	JPLS	•+3		IF EQUAL, THEN RESET PAGE
	ARX0'F			COUNT AND RETURN
	ARMD		PAGE	


```

MDIR      NPAGE
JUMP      .
MDIR      INULL
PAGE1     [PAGE TABLE
PAGE2
[PAGE3
[PAGE4
[CA LIST 0F TEXT T0 WRITE 0N THE SC0PE
P(11,26,13,50);P(23,37,I..N.);P(T..E..R..A.);P(C..T..I..V..)
P(E..11,40,0);P(13,60,L..0.);P(G..I..C..0);P(11,45,13,70)
P(L..A..B..0.);P(R..A..T..0.);P(R..Y..11,20);P(13,70,22,A..)
P(40,G..0,0)
P(R..A..P..H.);P(I..C..S..40);P(P..R..0..G.);P(R..A..M..40)
P(F..0..R..40);P(T..H..E..40);P(S..T..U..D.);P(Y..11,24,13)
P(74,0,F..40)
P(E..L..E..M.);P(E..N..T..A.);P(R..Y..40,L.);P(0..G..I..C..)
P(40,C..I..R.);P(C..U..I..T.);P(S..O..O)
P(11,70,13,80);P(B..Y..40,R.);P(56,40,L..56);P(40,J..0..H..)
P(N..S..0..N.);P(11,76,13,84);P(J..U..N..E.);P(40,61,71,67)
P(62,11,26,13);P(94,22,P..R.);P(E..S..S..40);P(F..U..N..C..)
P(T..I..0..N.);P(40,S..W..I.);P(T..C..H..40);P(61,40,T..0..)
P(40,R..E..T.);P(U..R..N..56);P(40,40,A..N.);P(Y..40,0,T..)
P(H..E..R..40);P(S..W..I..T.);P(C..H..40,T.);P(0..40,C..0..)
P(N..T..I..N.);P(J..E..56,0,1)
P(11,26,13,50);P(T..H..I..S.);P(40,P..R..0.);P(G..R..A..M..)
P(40,H..A..S.);P(40,F..0..U.);P(R..40,D..I.);P(S..T..I..N..)
P(C..T..40,M.);P(0..D..E..S.);P(11,36,13,60);P(D..E..F..I..)
P(N..E..22,55);P(55,P..R..0.);P(V..I..D..E.);P(S..40,0,0,1)
JUMP      .
MDAR'I    IRITE
MDAE'N    C1
ARM0      77735
MD10'0'L; 1400'H
ARX0'F
ARM0
MDAR'F    ITFLG
          IE9LR

```


IRITE CONTINUED

77736
-60000JH
-14
10
ITFLG
-1
-14
IRITE

ARM
MD10'A'L,
MDIC'A'L,
MDIC'0'L,
MDAR'N
JPN
MDIC'A'L,
MDIR'X

RETURN TO CALL

CONSTANTS AND VARIABLES

IXFLG:0
DUNF:0
PAGE:0
MASK1:7777
ISAV:0
ITFLG:0
IFSAY:0
IFCNT:0
IFSET:2
PLIM:2
C1:1

USER DEFINED INSTRUCTIONS

MD05=25000JH
MD06=26000JH
MD07=27000JH
MD10=30000JH
MD11=31000JH
AR07=27100JH

END OF INTRO (INTRODUCTION)

TERMINATE


```

EXPUNGE
TITLE LIL      [THIS PROGRAM IMPLEMENTS MAIN DISPLAY MODE
ENTRY LIL,DRDEF,LUPE,FCLER,E9LER,CADD, FCEND,DDEF
ENTRY DDEF2,CTAB,DR2,DRLAB,DRW8N,I1,1DRL,2DRL,TRITE
ENTRY ATELR,ARR8W,TWITE,NTAB
ENTRY DR3,ARG,ARG2,DR4,1DR4,2DR4,3DR4,DF1FG
MACR81  P(A1,A2,A3,A4,A5)
        A1VB/K + A2VB +A5VB/K + A3VB/K + A4VB + 0
ENDM
ZZ7=101      [ASCII CODE DEFNS...
IREPEAT ZZ, (A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z)
        ZZ\.=ZZZ
ZZZ=ZZZ+1
ENDI
P.=120

LIL:          ARX8'F
              ARMD
DDEF:          ARX8'F
              ARMD
DDEF2:         MDAR'F
              ARMD
              ARX8'F
              ARMD
              ARMD
              ARMD
              MDAR
              JPLS
              N98P
              MD10'8'L,
              MDAR'F
              ARMD
              ARX8'F
              ARMD
              MD10'A'L,

              [CLEAR FLAG T9 SKIP
              [DRAWING INITIALLY
              [ZERO MODE--MAIN DISPLAY
              [LOAD F/C PIV8T

              [CLEAR REFRESH FLAGS

              [CLEAR ALL DONE FLAG (FCLER)
              [T9 SKIP COMMANDS IN A MODE
              [WRITES COMMAND DISPLAY

              [SET UP L/P PIV8T

              [SHUT OFF AVG-1

TEXTW:

```


MDIC'A'L,	77763	[SET IC BITS FOR LCG-1
MDAR'F	TB1-1	
ARMD	77735	
MDAR'F	TE9LR	
ARMD	77736	
MDIC'0'L,	10	[TURN ON LCG-1
MDAR'N	TFLG	
JPAN	•-1	[WAIT TO DRAW COMMANDS
MDIC'A'L,	-14	
MDAR	DF1FG	[SKIP JUMP TO DRAW
JPAN	DRW	[IF THIS IS FIRST TIME
ARMD'0	FLG2	
JUMP	ADHNG	
MD10'0'L,	61400VH	[THIS DRAWS THE CIRCUIT,
UPSR	DRDEF	[LABELS, VALUES, ETC.
MDAR'N	ADFLG	[WAIT FOR A REFRESH
JPAN	•-1	
MDAR	MODE	[SELECT PROPER PROGRAM
JPLS	•+2	[TO RETURN TO BASED ON
JUMP	DDEF+2	[THE VALUE OF MODE
MDX0'F	1	
JPLS	•+2	
JUMP	\$STAR	[RETURN TO DEFINE (LOGMM)
MDX0'F	1'2	
JPLS	•+2	
JUMP	\$C9NCT	[RETURN TO CONNECT (CONCT)
MDX0'F	2'3	
JPLS	•+2	
JUMP	\$ANAL	[RETURN TO ANALYSIS (ANALR)
JUMP	\$INTRO	[RETURN TO INTRODUCTION
JUMP	•	[THIS SUBROUTINE DOES
JUMP	\$TBL0	[THE DRAWING
MDAR	77777	
ARAR'A'L,	\$TBL0	[SEE IF TBL0 IS EMPTY
MDX0'F	•+2	
JPLS	DRFFV	[IF S9- DRAW F-F VALUES
JUMP		
DRW:		
ADHNG:		
JUMPr:		
DRDEF:		

MDAR'F	\$TBL0	[SET PTR TO TOP OF TBL0
ARM0	TEMP1	
MDAR'X'I	TEMP1	
MDAR'A'L	7	[GET NEXT TBL0 ENTRY
MDAE'N'L	1	[AND FIND GATE TYPE
MDAS'F	CADD	
ARM0	TEMP2	
MDAR'I	TEMP1	
MDAR'A'L	7777677770	[GET DISPLAY COORDS OF
ARC7'F		[THIS GATE
MD11	NSCAL	
MD06	INTY	
MDAR'I	TEMP2	
MDAE'L	0	[LOAD E0V PIV0T WITH
MD05	77756	[PROPER DRAW COMMAND
ARM0	E0LER	[LOAD E0L PIV0T
MDAR'F	77757	
ARM0		
ARX0'F		
ARM0	DFLG	[THIS INST STARTS DRAW
MDIR	77756	
MDAR'N	DFLG	[WAIT TO FINISH
JPAN	•-1	
MDAR	TEMP1	
MDX0	\$TBL0	[IS TBL0 COMPLETELY DRAWN
JPLS	LUPE	[GET NEXT GATE IF NOT
MDAR'F	\$FFDT	[THIS ROUTINE DRAWS THE
ARM0	TEMP1	[FLIP-FL0P VALUES EITHER
MDAR'X	TEMP1	[1, 0,,0R (QUESTION MARK)
MDX0	\$FFDT	
JPLS	•+2	
JUMP	DRC0N	[JUMP TO DRAW CONNECTIONS
MDAR'H'I	TEMP1	[IF F-F VALUES ALL DRAWN
MDAR'A	C1	
JPLS	•+2	
JUMP	GET01	

LUPE:

DRFFV:

D-VRT:

MDAR'F	URDFF	[SETS UP UNDEF VALUE
ARM	FARG	[FOR FLIP-FLOPS
JUMP	SETXY	
MDAR'I	TEMP1	
MDAR'A	C7	
MDAS'F	FF9UT	
MDAR'A	MASK1	
ARM	FARG	[SETS UP 0 OR 1 DRAW
MDAR'I	TEMP1	[GET COORDS FOR VALUE
MDAR'A'L'	-100007	[DISPLAY
MDAE'L'	125077427	[ADD A FUDGE FACTOR
AR07'F		
ARX0'F		
ARM	TFLG	[TEXT-WRITE THIS VALUE
JPSR	TRITE	
0	0	
JUMP	DFVRT	[TO DRAW THE LINES
MD10'0'L'	614000H	[ENTERED IN THE CONNECT
ARX0'F		[MODE
ARM	DFLG	
MD06	INTY	
MD07	\$HXY1	
MD11	HSCAL	
MDAR'F	E0LER	
ARM	77757	
MDAR'L		
MD05	\$DATA1	[LOAD THE EV PIVOT
ARM	77756	
MDIR	77756	[DRAW THE CONNECTIONS
MDAR'N	DFLG	
JPAN	•-1	
MDAR	MODE	
MDX0'L'	2	
JPLS	DRW8N	[SKIP CURSOR DRAW AND
ARX0'F		[SELECTED VECTOR REDRAW
ARM	DFLG	[IF NOT IN CONNECT MODE

MDAR	\$C11	[REDRAW SELECTED LINE
JPLS	DRCUR	
MDAR'L		
MD05	\$DATA2	
ARM0	77756	
MDIR	77756	
MDAR'N	DFLG	
JPAN	•-1	
MD07	\$HDX	
ARX0'F		
ARM0	DFLG	
MDAR'L		
MD05	\$DATA	[DRAW THE CROSSHAIRS
ARM0	77756	[CONNECT CURS0R
MDIR	77756	
MDAR'N	DFLG	
JPAN	•-1	
ARM0'0	FLG2	[EXIT FROM THE DRAW MODE
MDIR	DRDEF	
MDAR	\$ANFLG	[THE LIL PART 0F THE
JPLS	•+2	[FOREGROUND/BACKGROUND
JUMP	DUNDR	[OPERATION
MDX0'F	1	
JPLS	DR2	[ANFLG=0 0R 1, D0 NOTHING
JUMP	DUNDR	
MDX0'F	1'2	
JPLS	DR4	[IF ANFLG = 2 THEN SET
MDAR'N	\$CFLAG	[UP THE PROPER VARIABLES
JPAN	DRLAB	[TO ACCEPT CHARACTERS
ARM0	\$CFLAG	[JUMP 0UT IF N0 LABELS YET
MDAR	\$CCNT	[SNEAKY CLEARING 0F FLAG
ARM0	T1ADD	
MDAR'X	\$CCNT	[SAVE THIS CARCOUNT TEMP
MDAR	\$CHAR	
MDX0'F	15	[IS THIS A C/R
JPLS	DD1	

DRCUR:

DUNDR:

DRW0N:

DR2:

MDAR	\$CCNT	[IF THIS IS THE FIRST
MDX0'F	1	[CHAR DEN T ACCEPT A
JPLS	DD2	[CARRIAGE RETURN
ARX0'F		
ARMD	\$CCNT	
JUMP	DRLAB	
MDAR	T1ADD	[SELECT THE PROPER A/N FIELD
MDAS'F	T1	[T0 STUFF THIS CHAR INTO
MDAR'A	MASK1	
MDAE'L	0	
MDAR		
ARIR'F	0	[ORIENT THIS CHAR FOR THE
MDAE'L	DR2LS	[PROPER A/N FIELD
ARLS	\$CHAR	
ARMD		
MDAR		
0		
N99P	DISL	[STORE SHIFTED CHARACTER
ARMD	\$CMASK	
MDAR	DR2LS	[SHIFT MASK T0 CHAR POSIT
MDIR		
ARAR'N	CTAB	[MASK OUT PREVIOUS CHARS
MDAR'A'I	DISL	
MDAR'0	CTAB	[RESTORE NEW CHAR IN CTAB
ARMD'I	\$CCNT	
MDAR	3	[TEST IF LAST CHAR FOR
MDX0'F	DRLAB	[THIS LABEL
JPLS	SIZIT	[SET SIZE BITS
MDAR'H	CTAB	[AND E0L IN NEXT TABLE
ARMD'X'I	CTAB	[ENTRY
MDAR	CTAB+2	
MDAS'N'F	\$URD	
MDAS'F	TURD	[SET THE END-9F-LIST BIT
ARMD	TURD	[IN THIS URD ENTRY (CHECKED)
MDAR'I	C1	
MDAR'0'H		

DD1:

DR2LS:

DD2:

ARM'D'I	TURD	[RESTORE URD
MDAR'X	TURD	
MDXØ	\$BURD	[TEST END 9F URD TABLE
JPLS	•+2	
JUMP	DR3	[JUMP TO SET UP FOR NUMBS
ARXØ'F	\$CCNT	[IF LABEL BUILD COMPLETE
ARM'D	DRLAB	[EXIT DRAW TO GET ANOTHER
JUMP	\$BURD	[LABEL NEXT TIME
MDAR'F	C1	
MDAE'N	MASK1	[SET UP POINTERS IN URD
MDAR'A	URLP	
ARM'D		[ARROW DRAW FLAG
ARXØ'F	ARØFG	
ARM'D	\$ANFLG	
MDAR	2	[ANFLG=2...WRITE-INPUT LABELS
MDXØ'F	WIN	
JPLS	TWITE	
JPSR	ILAB	
	1DRL	
JUMP	2'4	[ANFLG=4...INPUT NUMBS MSG
MDXØ'F	WANS	
JPLS	TWITE	
JPSR	INUM	
	1DRL	
JUMP	4'6	[ANFLG=6...ANSWER MESSAGE
MDXØ'F	1DRL	
JPLS	TWITE	
JPSR	ANSR	
	URLP	[GET NEXT URD ENTRY
MDAR'X	\$BURD	
MDXØ	DWAGN	[SEE IF LAST ENTRY
JPLS	DUNDR	[EXIT DRAWING IF SØ
JUMP	URLP	
MDAR'I'H	C1	[CHECK EØL 9F THIS URD
MDAR'A	2DRL	[SKIP ACCEPT INFO IF SET
JPLS		

DRLAB:

WIN:

WANS:

1DRL:

DWAGN:

MDXG	CTAB	[CHECK IF AT END OF CTAB
JPLS	•+2	
JUMP	DUNDR	[DRAW COMPLETE IF S0
JUMP	1DRL	[ELSE RETURN
ARXD'0	URFLG	[THIS ROUTINE ENTERED WHEN
MDAR'F	\$URD	[ALL THE LABELS HAVE BEEN
MDAE'N	C1	[DEFINED. SETS UP FOR
MDAR'A	MASK1	[ACCEPTING NUMBERS
ARMD	DR3P	
MDAR'X	DR3P	[THIS SEGMENT CLEARS OUT
MDX0	\$BURD	[THE EOL BIT SET IN THE
JPLS	•+2	[URD ENTRY WHEN LABELED
JUMP	DR3E	
MDAR'I	DR3P	[NOW USE EOL TO SHOW
MDAR'A	C1	[A VALUE HAS BEEN ASSIGNED
JPLS	DR3R	
MDAR'I'H	DR3P	
MDAR'A'L'	-1	
ARMD'I'H	DR3P	
JUMP	DR3R	
ARXD'F	UNFLG	[SET NUMBER FLAG
ARMD	\$ANFLG	[INCREMENT ANFLG TO 4
MDAR'X	\$RSTFG	[CHECK RESET (FNS-4
MDAR	\$ANCHK	[START ACCEPT NUMBS IF SET
JPAN	DUNDR	[ELSE EXIT DRAWING
JUMP	2'4	
MDX0'F	DR5	
JPLS	\$CFLAG	[THIS SEGMENT SIMILAR TO
MDAR'N	DRLAB	[DR2 EXCEPT WE ARE NOW
JPAN	\$CFLAG	[ACCEPTING NUMBERS
ARMD	\$CCNT	
MDAR	T1ADD	
ARMD	\$CCNT	
MDAR'X	\$CHAR	
MDAR	15	
MDX0'F		

1DR4:	JPLS	1DR4	[ONLY ACCEPT BINARY INPUTS
	JUMP	4DR4	
	MDAR	\$CHAR	
	MDX0'F	60	
	JPLS	*+2	
	JUMP	2DR4	
	MDX0'F	60'61	[ONLY ACCEPT BINARY INPUTS
	JPLS	N9G8	
	MDAR	T1ADD	
	MDAS'F	T1	
	MDAR'A	MASK1	
	MDAE'L		
	MDAR	0	[GET NUMERICAL CHARACTER
	ARIR'F		
	MDAE'L	0	
	ARLS	DR4LS	
	ARMQ	\$CHAR	
	MDAR		
	0		[SHIFT IT TO PROPER A/N FIELD
DR4LS:	N99P		
	ARMQ	INNUM	
	MDAR	\$CMASK	
	MDIR	DR4LS	
	N99P		
	ARAR'N		
	MDAR'A'I	NTAB	[MASK THIS DISPLAY NUMBER
	MDAR'0	INNUM	
	ARMQ'I	NTAB	
	MDAR	TNUMB	
	ARLS	1	
	N99P		
	ARMQ	TNUMB	
	MDAR	\$CHAR	[TNUMB IS THE ACCUMULATED
	MDAR'A	C1	[VALUE IN BINARY FORM
	MDAR'0		[SHIFT IT LEFT ONE PLACE
	ARMQ		[TO ALLOW NEW INPUT
	MDAR		[INT0 LEAST SIG. DIGIT(LSD)
	MDAR'A		[GET BINARY VALUE OF THIS
	MDAR'0		[DISPLAY CHARACTER AND
	ARMQ		[STORE IT IN TNUMB LSD


```

MDAR
MDX0'F
JPLS
MDAR
MDAS'N'F
MDAS'F
MDAR'A
ARMD
MDAR'F
MDAE'N
MDAR'A
ARMD
MDAR'X
MDX0
JPLS
JUMP
MDAR'I
JPAN
ARAR'H
MDAR'A
MDX0
JPLS
MDAR'I
MDAR'A
MDAE'L
ARMD
ARMD
MDAR
0
JUMP
MDAR'H
ARMD'X'I
MDAR
MDAS'N'F
MDAS'F
ARMD

```

```

$CCNT
3
DRLAB
NTAB
NTAB+1
$URD
MASK1
URDAD
$URI
C1
MASK1
URIPT
URIPT
$BTURI
*+2
4DR4
URIPT
3DR4
MASK1
URDAD
3DR4
URIPT
MASK1
0
*+2
TNUMB
3DR4
SIZIT
NTAB
NTAB
NTAB+2
$URD
TURD

```

3DR4:

4DR4:

MDAR'I	TURD	[SET EOL (CHECKED)
MDAR'0'H	C1	
ARMD'I	TURD	
MDAR'X	TURD	
MDAE	\$NOUTS	[ADD IN NUMBER OF
MDAR'A	MASK1	[CIRCUIT OUTPUTS
MDX9	\$BURD	
JPLS	•+2	[CHECK END OF URD
MDAR'X	\$ANFLG	[IF END INCREMENT ANFLG
ARX0'F		[TO CALCULATE RESPONSE ON
ARMD	\$CCNT	[NEXT PASS
ARMD	TNUMB	[ELSE RETURN TO GET NEXT NUMB
JUMP	DRLAB	
MDAR	\$CCNT	[PATCHES UP CCNT FOR NON-
MDAE'N	C1	[BINARY ENTRY AND
ARMD	\$CCNT	
JUMP	DRLAB	[RETURNS FOR NEXT NUMBER
MDX0'F	4'5	[ANFLG = 5•• THE RESPONSE
JPLS	DR6	[HAS BEEN CALCULATED•
MDAR'F	\$URI	[THIS SEGMENT FINDS EACH
MDAE'N	C1	[CIRCUIT OUTPUT (BIT 0 IS
MDAR'A	MASK1	[SET IN URI)
ARMD	DR5PT	
ARX0'F		
ARMD	NDFLD	[LOOK AT NEXT URI ENTRY
MDAR'X	DR5PT	
MDX9	\$BTURI	
JPLS	•+3	
MDAR'X	\$ANFLG	[CALL SEARCHED INCR ANFLG
JUMP	DRLAB	
MDAR'I	DR5PT	
JPAN	•+2	
JUMP	1DR5	[NOT CIRCUIT OUTPUT GET NEXT
MDAR'A	MASK1	[CIRCUIT OUTPUT FOUND
MDAE'L		
MDAR	0	

NEG0:	
DR5:	
1DR5:	

2DR5:	ARIR'F	DNUMB	[GET VALUE FROM ANSW
	ARMD	T1+2	[BASED ON URI POINTER
	MDAR'F	NDFLD	[THIS SEGMENT CONVERTS
	MDAE'N	MASK1	[EACH BINARY DIGIT IN THE
	MDAR'A		[ANSWER INTO DISPLAY FORM
	MDAE'L	0	
	MDAR		
	ARIR'F		
	MDAE'L	0	
	ARLS	DR5LS	
	ARMD	DNUMB	
	MDAR	NDFLD	
	ARRS'I		
	N99P	C1	[DISPLAY A ZERO OR ONE
	MDAR'A	60	[FOR THIS DIGIT
	MDAR'0'L;		[THIS SEGMENT ORIENTS THE
DR5LS:	0		[DISPLAY CHAR FOUND ABOVE
	N99P		[INT0 THE PROPER NTAB FIELD
	ARMD	INNUM	
	MDAR	\$CMASK	
	MDIR	DR5LS	
	N99P		
	ARAR'N		
	MDAR'A'I	NTAB	
	MDAR'0	INNUM	
	ARMD'I	NTAB	
	MDAR'X	NDFLD	
	MDX0'F	3	
	JPLS	2DR5	
	MDAR'X	NTAB	
	JUMP	1DR5-2	
DR6:	MDX0'F	5'6	[NOTHING TO DO HERE BUT
	JPLS	.	[DRAW ACCUMULATED TABLES
	JUMP	DRLAB	[TRAP IF ANFLAG GREATER
	JUMP	0	[THAN SIX
E0LER:	ARMD	ASAV	[END OF LIST HANDLER

TEQLR:	ARM'D	DFLG	[TEQLR JUSTS SETS A FLAG
	MDAR	ASAV	
	JUMP'I	EQLR	
	JUMP	O	[TEXT ENQ OF LIST HANDLER
	ARM'D	TSAV	
	MD10'A'L;	77757JH	[TURN OFF L/P
	ARM'D	TFLG	[SET THE TEXT FLAG WE ARE
	MDAR	TSAV	[CHANGING 9N AND RETURN
	JUMP'I	TEQLR	
	NQSP		[LIGHT PEN HANDLER
	ARM'D	LSAV	
	MDAR'L;	1	[DETERMINES WHICH COMMAND
	ARM'D	MDE	[WAS BEING DRAWN WHEN
	MDAR	77735	[LIGHT PEN INTERRUPT
	MDAR'A'L;	77777	[OCCURED AND SETS MDE
	ARAR'N'F		[TO APPROPRIATE VALUE
	MDAS'F		
	JPAN	TB2	
	JUMP	•+2	[EXIT--MDE=1
	MDAR'X	LPEND	
	MDAR	MDE	
	MDAR'A'L;	77735	
	ARAR'N'F	77777	
	MDAS'F		
	JPAN	TB3	
	JUMP	•+2	[EXIT--MDE=2
	MDAR'X	LPEND	
	MDAR	MDE	
	MDAR'A	77735	
	ARAR'N'F	MASK1	
	MDAS'F		
	JPAN	TB4	
	JUMP	•+2	[EXIT--MDE=3
	MDAR'X	LPEND	[EXIT--MDE=4
	MDAR	MDE	[MDE SET ••RETURN AFTER
	MDAR'A	LSAV	[CLEARING INTERRUPT
	ARAR'N'F	LPHLR	
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		
	MDAR		
	MDAR'A		
	ARAR'N'F		
	MDAS'F		
	JPAN		
	JUMP		
	MDAR'X		

FCFLER:

JUMP
FPRI
ARMD
MDAR'X
MDAR'N
JPAN
MDAR
MDX0
JPLS
ARMD'0
ARX0'F
ARMD
MDAR
UPRI
JUMP'I
JUMP
MDAR'I
MDAE'N
ARMD
MD10'0'L;
ARX0'F
ARMD
ARMD
MDAR'F
ARMD
MD05
MDAR'N
JPAN
MDAR'F
ARMD
MD10'A'L;
MDIC'A'L;
MDIC'0'L;
MDAR'N
JPAN

0

BSAV
FCNT
FLG2
FCEND
FSET
FCNT
FCEND
ADFLG

FCNT
BSAV

FCFLER

•
TRITE
C1
77735
61400JH

DFLG
TFLG
E0LER
77757
M0VIT
DFLG
•-1
ATELR
77736
17777JH
77763
14
TFLG
•-1

FCEND:

TRITE:

[FRAME CLOCK HANDLER

[REFRESHES EVERY FSET TICKS

[IF ALL FIGURES DRAWN

[AND PROPER N9. 0F TICKS

[SET ALL DONE FLAG (REFRESH)

[UNFREEZE INTERRUPTS,CLEAR
[THIS INTERRUPT AND RETURN
[A SUBROUTINE FOR WRITING
[TEXT BLOCKS USING AVG1
[OUTPUT FOR X Y POSITION
[ASSUMES XY DESIRED IS IN
[DESTINATION 7

[TURN ON IC 26 AND 27

MDIC'A'L;
MDIR'X
JUMP
MDAR'I
MDAE'N
ARMD
MD10'0'L;
ARX0'F
ARMD
MDAR'F
ARMD
MD10'A'L;
MDIC'A'L;
MDIC'0'L;
MDAR'N
JPAN
MDIC'A'L;
MDIR'X
JUMP
ARMD
ARMD'0
MDAR
JUMP'I

TWITE:

-14
TRITE
.
TWITE
C1
77735
1400VH
TFLG
ATELR
77736
-60000VH
-14
10
TFLG
.-1
-14
TWITE
.
TSAV
TFLG
TSAV
ATELR

ATELR:

[CONSTANTS AND VARIABLES]

FCNT:0
FSET:2
FLG2:0
DFLG:0
ADFLG:0
TFLG:0
TEMP1:0
TEMP2:0
ASAV:0
BSAV:0

[RESET LCG1 AND
[RETURN TO CALL
[WRITES TEXT BLOCKS
[SIMILAR TO TRITE BUT
[ASSUMES XY POSITION
[IS INCLUDED IN TEXT

[IC 27 NOT TURNED ON..
[NOT USING AVG1 OUTPUT
[FOR XY POSITIONING

[TEXT END OF LIST HANDLER

[MORE CONSTANTS AND VARIABLES...]

TSAY:0
LSAY:0
MODE:0
NSCAL:5770JH
HSCAL:37777JH
INTY:20000

C1:1
MASK1:77777
URLP:0
FUDGE:2200
FUDGY:0
FUDGN:600
URFLG:0
TURD:0
T1ADD:
DK3P:0
TNUMB:0
URDAD:0
URIPT:0
UNFLG:0
CP9S:0
ARCFG:0
NDFLD:0
DNUMB:0
DREPT:0
INNUM:0
DISL:0
C7:7
DF1FG:0
URCFF:
FFOUT:

0

P(37,0,0,77,1) [UNDEFINED F-F VALUE
[ZER9(RESET)F-F VALUE
[9NE(SET)F-F VALUE

TABLE OF DATA SETS FOR
DRAWING THESE GATES..
LOCATED IN LOGMM

\$DADD
\$DORD
\$DINVD
\$FFTD
\$FFSD
\$FFKD

CADD:

BIT SHIFT TABLE FOR DISPLAY

T1: 16.
8.
1.

CHARACTER DISPLAY TABLE
PRE-LOADED WITH SIZE AND
END OF LIST

CTAB: 1100100000 10.
REPEAT
1100100000
ENDR

NUMBER DISPLAY TABLE
SIMILARLY PRE-LOADED

NTAB: 1100100000 10.
REPEAT
1100100000
ENDR

TEXT BLOCKS TO DRAW
COMMAND WORDS

TB1: P(11,24,13,90.)
P(23,23,37,0)
P(D,,E,,F,,I.)
P(N,,E,,O,,O)
P(11,56,0,0)
P(C,,8,,N,,N.)
P(F,,C,,T,,O)
P(11,80,0,0)
P(A,,N,,A,,L.)
P(Y,,Z,,E,,O.)
P(11,54,13,96.)
P(I,,N,,T,,R.)
P(8,,D,,U,,C.)
P(T,,I,,8,,N.)
P(O,0,0,0,1)

TB2:

TB3:

TB4:

M0VIT:
 SIZIT:
 ARROW:
 ILAB:

0000100000
 11001
 P(55,55,76,0,1)
 P(11,56,13,90,)
 P(I,N,P,U,)
 P(T,11,72,L,)
 P(A,B,E,L,)
 P(S,0,0,0,1)
 P(11,56,13,90,)
 P(I,N,P,U,)
 P(T,11,72,0)
 P(V,A,L,U,)
 P(E,S,0,0,1)
 P(11,56,13,90,)
 P(23,A,N,S,)
 P(W,E,R,0,1)

INUM:

ANSR:

MD05=250000VH
 MD06=260000VH
 MD07=270000VH
 MD10=300000VH
 MD11=310000VH
 AR07=27100VH
 TERMINATE

[USER DEFINED COMMANDS

[A DUMMY MOVE COMMAND
 [ARROW TEXT DATA
 [VARIOUS MESSAGE TEXTS

[END OF LIL (MAIN DISPLAY MODE).....

LIST OF REFERENCES

1. Educational Aids in Engineering, American Society for Engineering Education, March 1955.
2. Hauer, J. F. and Syms, G. H., "Digital Machine Simulation Programs in Electrical Engineering Education," Proceedings of Purdue 1971 Symposium on Applications of Computers to Electrical Engineering Education, p. 155-163, 26-28 April 1971.
3. Johnson, C. I., "Principles of Interactive Systems," IBM Systems Journal, v. 7, p. 147-173, 1968.
4. Licklider, J. C. R., "Man-Computer Symbiosis," IEEE Transactions on Human Factors, HFE-1, p. 4-11, March 1960.
5. Coons, S. A., "An Outline of the Requirements for a Computer-Aided Design System," AFIPS Conference Proceedings, Spring Joint Computer Conference, p. 299-304, 1962.
6. Sutherland, I. E., "Sketchpad - A Man-Machine Graphical Communication System," AFIPS Conference Proceedings, Spring Joint Computer Conference, p. 329-346, 1962.
7. IBM Report 1620-EE-02X, 1620 Electronic Circuit Analysis Program (ECAP), 1965.
8. Schroer, B. J., "Expanded Capabilities of CSMP Graphics of the IBM 1130 Digital Computer," Simulation, v. 14, p. 205-214, May 1970.
9. Katzenelson, J., "AEDNET: A Simulator for Nonlinear Networks," Proceedings of the IEEE, v. 54, p. 1536-1522, November 1966.
10. Dertouzos, M. L., "CIRCAL: On-Line Circuit Design," Proceedings of the IEEE, v. 55, p. 637-654, May 1967.
11. Dertouzos, M. L., "An Introduction to On-Line Circuit Design," Proceedings of the IEEE, v. 55, p. 1961-1971, November 1967.
12. Skinner, B. F., The Technology of Teaching, Appleton-Century-Crofts, 1968.
13. Silvern, G. M. and L. C., "Programmed Instruction and Computer-Assisted Instruction--An Overview," Proceedings of the IEEE, v. 54, p. 1648-1656, December 1966.
14. Shannon, C. E., "A Symbolic Analysis of Relay and Switching Circuits," Transactions of the AIEE, v. 57, 1938.
15. Adage Graphics Terminal, Systems Reference Manual, Adage Inc., Boston, Mass.

16. Baskin, H. B. and Morse, S. P., "A Multilevel Modeling Structure for Interactive Graphic Design," IBM Systems Journal, v. 7, p. 218-228, 1968.
17. Williams, R., "On the Application of Graph Theory to Computer Data Structures," Advanced Computer Graphics, Economics, Techniques and Applications, Green, R. E. and Parslow, R. D., eds., p. 775-801, Plenum Press, 1971.
18. Busacker, R. G. and Saaty, T. L., Finite Graphs and Networks: An Introduction with Applications, McGraw-Hill, 1965.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2. Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3. Assoc Professor G. A. Rahe, Code 52 Ra Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
4. LT Robert Lee Johnson, Jr., USN 26 Tremont Street Penacook, New Hampshire 03301	1

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Naval Postgraduate School Monterey, California 93940		2a. REPORT SECURITY CLASSIFICATION Unclassified	
		2b. GROUP	
3. REPORT TITLE Interactive Logic Laboratory			
4. DESCRIPTIVE NOTES (Type of report and, inclusive dates) Master's Thesis; June 1972			
5. AUTHOR(S) (First name, middle initial, last name) Robert Lee Johnson, Jr., Lieutenant, United States Navy			
6. REPORT DATE June 1972		7a. TOTAL NO. OF PAGES 128	7b. NO. OF REFS 18
8a. CONTRACT OR GRANT NO.		9a. ORIGINATOR'S REPORT NUMBER(S)	
b. PROJECT NO.			
c.		9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
d.			
10. DISTRIBUTION STATEMENT Approved for public release; distribution unlimited.			
11. SUPPLEMENTARY NOTES		12. SPONSORING MILITARY ACTIVITY Naval Postgraduate School Monterey, California 93940	
13. ABSTRACT <p>This thesis documents an investigation into the use of a computer graphics terminal to demonstrate the basic concepts of logical design. The areas of computer-assisted instruction, computer graphics, and computer-aided design are reviewed prior to the discussion of the creation of the INTERACTIVE LOGIC LABORATORY. The program is implemented on the Adage Graphics Terminal - 10 (AGT-10) of the Naval Postgraduate School Computer Laboratory.</p> <p>The main emphasis of the program discussion is on the degree of interaction achieved by the program and its possible use as a learning aid for students of basic logical design courses. A bipartite graph is used to depict the network topology of the logic circuit and the program is quite successful in the simulation of simple logic circuits.</p>			

KEY WORDS

LINK A

LINK B

LINK C

ROLE

WT

ROLE

WT

ROLE

WT

KEY WORDS

Interactive Computer Graphics

Computer-aided design

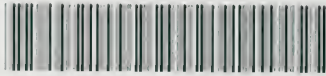
Computer-assisted instruction

Bipartite graph

Logic circuit design

thesJ628

Interactive logic laboratory.



3 2768 001 02691 7

DUDLEY KNOX LIBRARY